Multi-core mixed-criticality real-time scheduling

Ramanathan, Saravanan

2019

Ramanathan, S. (2019). Multi-core mixed-criticality real-time scheduling. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/82989>

<https://doi.org/10.32657/10220/47564>

MULTI-CORE MIXED-CRITICALITY REAL-TIME SCHEDULING

SARAVANAN RAMANATHAN

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University in partial fulfilment of the
requirement for the degree of Doctor of Philosophy

2019

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

24/1/2019
.....
Date

R.
.....
Saravanan Ramanathan

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

24/1/2019

.....
Date

Arvind

.....
Arvind Easwaran

Authorship Attribution Statement

This thesis contains material from 3 papers published in the following peer-reviewed conferences and journal where I was the first and/or corresponding author.

- Chapter 4 is published as S. Ramanathan, A. Easwaran, and H. Cho, “Multi-Rate Fluid Scheduling of Mixed-Criticality Systems on Multiprocessors”, in Real-Time Systems (RTS), vol. 54, no. 2, pp. 247-277, April 2018.

The contributions of the co-authors are as follows:

- I co-designed the problem with A/Prof Arvind Easwaran.
 - A/Prof Hyeonjoong Cho provided the initial project direction and helped in formulating the model.
 - I wrote the drafts of the manuscript. The manuscript was revised by A/Prof Arvind Easwaran.
 - All simulation experiments, including synthetic task set generation, was conducted by me in the Hardware and Embedded Systems Laboratory. I also analyzed the data.
 - A/Prof Arvind Easwaran assisted in designing experiments and provide guidance in the interpretation of simulation data.
- Chapter 5 is published as S. Ramanathan and A. Easwaran, “Utilization Difference Based Partitioned Scheduling of Mixed-Criticality Systems”, in Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, pp. 238-243.

The contributions of the co-authors are as follows:

- A/Prof Arvind Easwaran suggested the research area.
- I prepared the manuscript drafts. The manuscript was revised together with A/Prof Arvind Easwaran.
- I performed all the simulations, and conducted data evaluation.

- Chapter 6 is published as S. Ramanathan and A. Easwaran, “Mixed-Criticality Scheduling on Multiprocessors with Service Guarantees”, in IEEE International Symposium on Real-Time Computing (ISORC), 2018, pp. 17-24.

The contributions of the co-authors are as follows:

- A/Prof Arvind Easwaran assisted in the problem formulation and provide initial project directions.
- I performed all the experimental evaluation, and analyzed data. I also wrote the manuscript.

24/1/2019

.....
Date

R.

.....
Saravanan Ramanathan

Acknowledgements

Throughout the course of my Ph.D. I have received support and help from several people without which this thesis would not have been possible. I wish to express my sincere gratitude to every one of them.

First and foremost, I would like to thank my advisor Arvind Easwaran for his everlasting support during these past four years. Arvind introduced me to the world of research and showed me the joy of solving research problems. His meticulousness, myriad insightful discussions, and persistence have helped me identify and tackle several challenging obstructions during this research. I am thankful to him for giving me space and freedom to explore the fields I am interested in. He has been a great mentor who has taught me many things in real time (life) than just ‘real-time systems’. I still remember the time we spent during several conference travel, team outings, and hiking nature trails.

Besides my advisor, I would like to thank my thesis advisory committee members, Insik Shin and Alwen Fernando Tiu, for their valuable suggestions and encouragement that I received throughout the process. Insik has always been a great source of inspiration and gave me directions to the semi-partitioned scheduling work presented in Chapter 6. I am grateful to Cho Hyeonjoong for all the help received, during his stay here in Singapore, in formulating the multi-rate fluid model presented in Chapter 4. I am also grateful to Jaewoo Lee and Zhishan Guo for their interactions, guidance and collaboration in the fluid scheduling.

It has been a great pleasure to spend four years in the School of Computer Science and Engineering, NTU. I would like to express my appreciation to administrative and non-teaching staff in the department for their service and help during my study. I take this opportunity to extend my appreciation to Chue Ngee Tat, Hardware and Embedded Systems Lab (HESL), for setting up my lab environment, procuring hardware and providing all logistics support throughout my course of study. I want to thank all the past and present members of the Cyber-Physical Systems (CPS) research group from whom I have learned a lot. Especially, I want to thank my colleagues

Xiaozhe Gu, Bibin Chandrashekar, Mohammed Shihabul Haque for the fruitful discussions, mock presentations, proof-readings and collaborations.

I would like to thank all my friends in HESL, department and in the university, Kratika, Nimisha, Ankita, Kalindu, Tushar, Vijetha, Supriya, Abhishek, Queen and Laskhmi, to name a few. I extend my heartfelt appreciation to my badminton and karate buddies, Ajinkya, Chiam, Daniel, Sid, Petchiyappan, Nanda, Nehemiah, Bingyan, Graham, Darren, Jingyuan, Sisi and Vijitha for keeping me in good shape during these four years. The exciting and enduring time spent with you will always be remembered.

My life as a Ph.D. student has been less tormenting thanks to all my friends for their unconditional friendship and support. I especially want to thank my roommates Sriram, Padmanabhan and Vijay for sticking with me and dealing with my tantrums for the last four years. Life in Singapore would not have been so satisfying without my friends from the Masters programme. A big thanks to Shiva, Karthick, Prakash, Chandrasekar and Aung for keeping my non-academic life so lively and fun. Thanks to all my friends who accompanied me during my travel. I cherish the moments spent with Kavimani, Gaukhow, Nitin, Vaishnavi, Venkat and Rethina during conference travel.

A special thanks to Arvind, Abhiruchi, Vihan and Akira for making me feel at home, for all the delicious food and the good time spent together. I am indebted to you for sharing your knowledge and wisdom about life.

Finally, I would like to thank my parents, my sister, brother-in-law and my niece for their unconditional love and care. Thank you for wholeheartedly supporting my ambitions and giving freedom to achieve them. Your inspiration helped me to carry forward this work towards completion.

Abstract

Modern safety-critical real-time systems are realized via integration of multiple system components having varying criticality, designated based on consequence of failures, onto a single shared hardware platform. Such systems are known as ‘Mixed-Criticality Systems’. These systems are subjected to different certification standards that require high confidence (or assurance level against failures) for critical components (*e.g.*, flight control) and relatively low confidence for less/non-critical components (*e.g.*, cabin light control).

The correctness of a mixed-criticality system depends on the correct functional behavior of the components with their corresponding assurance level at different criticality levels. To guarantee correctness, conventional real-time scheduling techniques that statically reserve resources to the components can be used. However, conservative reservation at all criticality levels often leads to under-utilization of system resources as the components do not necessarily require them at all times. To address this problem, mixed-criticality scheduling model was proposed. The fundamental objective is to efficiently allocate resources while ensuring safe execution of the critical components. The mixed-criticality scheduling requirement is as follows: all components must be scheduled to meet their deadlines as long as sufficient processing capacity is available and once the system demand is increased, critical components must be prioritized to meet their deadlines, whereas less/non-critical components can be penalized to meet the system demand.

Although the problem of mixed-criticality scheduling on single-core platforms has been well studied, for multi-core platforms it has largely remained open. With this objective in focus, this thesis addresses some important problems of scheduling mixed-criticality systems on multi-core processors. Multi-core scheduling can broadly be classified as global and partitioned scheduling. Global scheduling allows components to execute on any core and migrate between cores at runtime. Partitioned scheduling statically assigns components to an individual core and does not allow them to migrate at runtime.

Most existing global scheduling algorithms for mixed-criticality systems suffer from poor schedulability. To address this problem, a fluid execution rate based scheduling algorithm with

a bounded analytical performance for mixed-criticality systems has been proposed earlier. However, the proposed fluid scheduling algorithm still suffer from schedulability issues. To this end, we first focus on the design of a global scheduling algorithm that is capable of scheduling a large class of such systems on multi-core platforms and has bounded analytical performance. We propose a multiple fluid execution rate based scheduling model for mixed-criticality systems and derive the analytical test for the algorithm. The proposed model considers different execution rates based on the system operation mode. Thus, by considering mode based dynamic execution rates the proposed algorithm is able to achieve better schedulability compared to existing algorithms. Furthermore, the proposed algorithm also has an optimal performance bound in terms of processor speedup.

Most studies in mixed-criticality scheduling make an unrealistic assumption that all the less critical components can be abandoned or discarded when the critical components demand for additional processing resources. However, this method seriously impacts the performance of the less critical components which may not be suitable for many practical systems that require minimum service guarantees for these components. To address this issue, we focus on the partitioned scheduling for mixed-criticality systems which is preferred in the industry due to its strong isolation (freedom from interference) feature. The components executing on a core are unaffected by the critical components executing on another core unlike in global scheduling. Partitioning scheduling comprises components-to-core mapping (partitioning) strategy and scheduling algorithm on each core. Existing studies use classical partitioning strategies which are not efficient for the mixed-criticality setting. To address this problem, we propose an efficient partitioning strategy that has good performance for a wide variety of single-core mixed-criticality scheduling algorithms. This technique distributes the additional demand of the critical components evenly among all processing cores. By evenly distributing it, the scheduling capability of each core is increased, thus increasing the overall schedulability.

To support maximum less critical components in the partitioned environment, we propose the semi-partitioned mixed-criticality scheduling model. In this model, the less critical components are migrated at runtime from a core which has insufficient processing capacity to another that has sufficient processing capacity. The resources corresponding to the migrating components are statically reserved on the core to which they migrate. Thus, by migrating these components at runtime, we are able to support maximum less critical components for a longer duration. These components will be eventually suspended from execution if the processing cores are not capable of supporting them.

Table of Contents

Acknowledgements	V
Abstract	VII
List of Figures	XIII
List of Tables	XV
List of Abbreviations	XVI
List of Notations	XVII
List of Publications	XX
1 Introduction	1
1.1 Real-Time Cyber-Physical Systems	1
1.1.1 Computation Platforms	2
1.1.2 Real-Time Scheduling Theory	2
1.2 Mixed-Criticality (MC) Systems	3
1.2.1 Mixed-Criticality Scheduling	4
1.3 Motivation and Problem Statement	6
1.4 Contributions and Organization	7
2 Background	10
2.1 Real-Time System Descriptions	10
2.1.1 Workload Model	10
2.1.2 Taxonomy of Real-Time Scheduling	11
2.1.3 Feasibility, Optimality and Speed-up Factor	12
2.1.4 Fluid Scheduling	13

TABLE OF CONTENTS	X
-------------------	---

2.2	MC System Descriptions	15
2.2.1	MC Workload Model	15
2.2.2	Execution Semantics	16
2.3	MC Scheduling Algorithms	18
2.3.1	Earliest Deadline First with Virtual Deadlines (EDF-VD) Algorithm . . .	19
2.3.2	Dual-rate Fluid Scheduling Algorithm	20
3	Related Work	27
3.1	Single-core MC Scheduling	27
3.2	Multi-core MC Scheduling	29
3.2.1	Global Scheduling	29
3.2.2	Partitioned Scheduling	30
3.2.3	Semi-Partitioned Scheduling	32
3.3	Improving the Execution of LC Tasks	33
3.3.1	Multi-core Techniques	33
4	Generalized and Efficient MC Fluid Scheduling Model	36
4.1	Introduction	37
4.2	System Description	38
4.2.1	Dual-rate fluid scheduling model and its limitation	38
4.3	Motivational Example	39
4.4	Multi-rate Fluid Scheduling Model	41
4.5	Schedulability Test	43
4.5.1	LC mode task schedulability test	43
4.5.2	LC mode platform feasibility test	44
4.5.3	HC mode platform feasibility test	44
4.5.4	HC mode task schedulability test	44
4.5.4.1	Schedulability test for stable HC mode jobs	45
4.5.4.2	Schedulability tests for carry-over jobs	45
4.5.4.3	Schedulability tests for transition HC mode jobs	47
4.6	Properties of the multi-rate model	51

TABLE OF CONTENTS	XI
-------------------	----

4.6.1	Dual-rate Generalization	51
4.6.2	Speed-up Optimality	53
4.7	Mapping to non-fluid platform	54
4.8	Multi-rate Assignment Strategy	57
4.9	Experimental Evaluation	60
4.9.1	Task Set Generation	60
4.9.2	Evaluation of Overall Schedulability Performance	61
4.9.3	Evaluation of MC Fluid Scheduling Algorithms	63
4.9.4	Performance Evaluation for Varying Task Parameters	65
4.10	Efficacy of the multi-rate fluid scheduling	69
4.10.1	Comparative evaluation: dual-rate vs multi-rate fluid model	70
4.10.2	Sub-optimality of the multi-rate fluid model	71
4.10.3	Challenges and considerations for implementation	73
4.11	Conclusions	74
5	Efficient Partitioning Strategy for MC Systems	76
5.1	Introduction	77
5.2	System Descriptions and Scheduling Algorithms	78
5.2.1	Single-core MC Scheduling Algorithms	78
5.3	Utilization Difference based Partitioning (UDP)	78
5.3.1	CA-UDP Strategy	79
5.3.2	CU-UDP Strategy	82
5.4	Evaluation	84
5.4.1	Task Set Generation	85
5.4.2	Evaluation of Implicit-deadline Algorithms with known Speed-up Factor	85
5.4.3	Evaluation of Implicit-deadline Algorithms w/o Speed-up Factor	87
5.4.4	Evaluation of Constrained-deadline Algorithms	89
5.4.5	Performance Evaluation for Varying Task Parameters	91
5.5	Conclusions	93
6	Semi-Partitioned Mixed-Criticality Scheduling	94

6.1	Introduction	95
6.2	System Description	95
6.2.1	Execution Semantics	96
6.3	Semi-Partitioned Mixed-Criticality (SPMC) Model	96
6.3.1	Bounding the Server Utilization	98
6.4	Schedulability Analysis	101
6.4.1	Low Criticality Mode	101
6.4.2	High Criticality Mode	104
6.4.3	Bounding the number of preemptions	107
6.5	Task Allocation Strategy for SPMC	108
6.6	Evaluation	109
6.6.1	Task Set Generation	110
6.6.2	Evaluation of Schedulability Performance	110
6.6.3	Support for LC tasks on Dual-core Systems	112
6.6.4	Performance Evaluation for Varying Task Parameters	113
6.6.5	Evaluation of Semi-Partitioned Scheduling	114
6.6.6	LC support for Varying Number of Mode Switch	115
6.7	Conclusions	116
7	Conclusions and Future Work	117
7.1	Conclusions	117
7.2	Future Research Directions	119
	References	122

List of Figures

1.1	DO-178B classification of criticality levels	4
2.1	Periodic task $\tau_i = (10, 3, 8)$	11
2.2	Fluid execution of a task τ_i	13
2.3	MC execution modes	17
2.4	Mode switch in multi-core MC scheduling	18
2.5	Mode switch in dual-rate fluid scheduling	21
2.6	The $\bar{f}'_i(X_i = x)$ plot for varying x	24
4.1	Schedule in the high-criticality mode	41
4.2	Worst-case mode switch instant of a carry-over job	45
4.3	A carry-over job that does not trigger a mode switch	46
4.4	Transition jobs	48
4.5	Evaluation of algorithms with known speed-up factor for varying m	63
4.6	Percentage of task sets that are schedulable under multi-rate but not under dual-rate	64
4.7	Evaluation of fluid scheduling algorithms for varying n_H	66
4.8	Evaluation of fluid scheduling algorithms for varying task parameters	68
4.9	Evaluation of fluid scheduling algorithms for varying task set parameter (U^L_E/m)	69
4.10	Schedule of multi-rate infeasible task system	72
5.1	Comparison of CA-UDP and CA-Wu-F	81
5.2	Comparison of CA-UDP and CU-UDP	83
5.3	Evaluation of implicit-deadline algorithms with known speed-up factor	86
5.4	Evaluation of implicit-deadline algorithms w/o speed-up factor	88

5.5	Evaluation of constrained-deadline algorithms	90
5.6	Varying Percentage of Task Criticality	91
6.1	SPMC Model	97
6.2	Evaluation of Schedulability Performance	111
6.3	LC Utilization Support for Dual-core Systems	112
6.4	Varying percentage of LC tasks	113
6.5	Percentage of WCR-EDF infeasible task sets that are schedulable by SPMC . . .	114
6.6	Varying number of the mode switch	115

List of Tables

2.1	Example implicit-deadline task system and its rate assignment.	22
2.2	Example 3-Criticality implicit-deadline task system scheduled on dual-core plat- forms	25
2.3	Rate assignment to the example 3-criticality task system	26
4.1	Example task system schedulable under the multi-rate fluid model, but not under the dual-rate fluid model on dual-core platforms. C_i^L denotes LC execution budget, C_i^H denotes HC execution budget, T_i denotes task period, $u_i^L = C_i^L/T_i$ and $u_i^H = C_i^H/T_i$. For the dual-rate fluid model, θ_i^L denotes exe- cution rate in the LC mode and θ_i^H denotes execution rate in the HC mode. For the multi-rate fluid model, θ_i^L denotes execution rate in the LC mode, $\theta_{i,j}^H (j \in$ $\{1, 2, 3\})$ and θ_i^H denote several execution rates for the HC mode, and $w_j (j \in$ $\{1, 2, 3\})$ denotes duration of time for which rate $\theta_{i,j}^H$ will be used by task τ_i . After $w_1 + w_2 + w_3$ time units in the HC mode, each HC task τ_i will use the rate θ_i^H	40
4.2	A feasible task system deemed unschedulable by the multi-rate model	72
5.1	Example task system scheduled on 2 cores using CA-UDP	81
5.2	Example task system scheduled on 2 cores using CU-UDP	83

List of Abbreviations

AMC	Adaptive Mixed-Criticality
CA	Criticality-Aware
CU	Criticality-Unaware
ECDF	Earliest Carry-over Deadline First
EDF-VD	Earliest Deadline First with Virtual Deadlines
HC	High-Criticality
LC	Low-Criticality
MC	Mixed-Criticality
OERA	Optimal Execution Rate Assignment
SOMA	Speed-up Optimal Multi-rate Assignment
SPMC	Semi-Partitioned Mixed-Criticality
UDP	Utilization Difference based Partitioning
WCET	Worst-Case Execution Time

List of Notations

τ_i	i^{th} task
T_i	Period of τ_i
C_i	Worst-case execution budget of τ_i
D_i	Relative deadline of τ_i
u_i	Utilization of τ_i
θ_i	Execution rate of τ_i
T_{min}	Smallest period of all tasks
m	Number of processing cores
χ_i	Criticality level of τ_i
C_i^L	LC execution budget of τ_i
C_i^H	HC execution budget of τ_i
D_i^L	Virtual deadline of τ_i in LC mode
x	Deadline scaling factor
u_i^L	LC utilization of τ_i
u_i^H	HC utilization of τ_i
θ_i^L	Execution rate of τ_i in LC mode
θ_i^H	Execution rate of τ_i in HC mode
$\theta_{i,j}^H$	Execution rate of τ_i in j^{th} transition window
η_i	Demand of τ_i in LC mode

τ_L	Set of LC tasks
τ_H	Set of HC tasks
n_H	Number of HC tasks
P_H	Percentage of HC tasks
P_L	Percentage of LC tasks
U_L^L	Total LC utilization of LC tasks
U_H^L	Total LC utilization of HC tasks
U_H^H	Total HC utilization of HC tasks
ϕ_k	k^{th} processing core
$\tau(\phi_k)$	Set of tasks assigned to k^{th} processing core
$U_L^L(\phi_k)$	Total LC utilization of LC tasks in k^{th} processing core
$U_H^L(\phi_k)$	Total LC utilization of HC tasks in k^{th} processing core
$U_H^H(\phi_k)$	Total HC utilization of HC tasks in k^{th} processing core
τ_{NM}	Set of non-migrating LC tasks
τ_{OM}	Set of outgoing migrating LC tasks
τ_{IM}	Set of incoming migrating LC tasks
$\tau_{OM}(\phi_k)$	Set of outgoing migrating LC tasks in k^{th} processing core
$\tau_{IM}(\phi_k)$	Set of incoming migrating LC tasks in k^{th} processing core
OM_k	Server corresponding to $\tau_{OM}(\phi_k)$
IM_k	Server corresponding to $\tau_{IM}(\phi_k)$
$U_{OM}(\phi_k)$	Total utilization of $\tau_{OM}(\phi_k)$
$U_{IM}(\phi_k)$	Total utilization of $\tau_{IM}(\phi_k)$
$U_s(\phi_k)$	Total server utilization in k^{th} processing core
U_B	Normalized utilization bound
U_B^L	Utilization bound for LC mode

U_B^H	Utilization bound for HC mode
S	Server period
α	Server inflation factor
δ	Server period scaling factor

List of Publications

- Saravanan Ramanathan and Arvind Easwaran, “Mixed-Criticality Scheduling on Multiprocessors with Service Guarantees”, in IEEE International Symposium on Real-Time Computing (ISORC), 2018, pp. 17-24.
- Saravanan Ramanathan, Arvind Easwaran and Hyeonjoong Cho, “Multi-Rate Fluid Scheduling of Mixed-Criticality Systems on Multiprocessors”, in Journal Real-Time Systems (RTS), vol. 54, no. 2, pp. 247-277, April 2018.
- Jaewoo Lee, Saravanan Ramanathan, Kiew-My Phan, Arvind Easwaran, Insik Shin and Insup Lee, “MC-Fluid: Multi-core Fluid-based Mixed-Criticality Scheduling”, IEEE Transactions on Computers, vol. 67, no. 4, pp. 469-483, 1 April 2018.
- Saravanan Ramanathan and Arvind Easwaran, “Utilization Difference Based Partitioned Scheduling of Mixed-Criticality Systems”, in Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, pp. 238-243.
- Saravanan Ramanathan and Arvind Easwaran, “Evaluation of Mixed-Criticality Scheduling Algorithms using a Fair Taskset Generator”, in International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS), 2016, pp. 19-24.
- Saravanan Ramanathan, Xiaozhe Gu and Arvind Easwaran, “The Feasibility Analysis of Mixed-Criticality Systems”, in Real-Time Scheduling Open Problems Seminar (RT-SOPS), 2016, pp. 5-6.
- Saravanan Ramanathan and Arvind Easwaran, “MC-Fluid: rate assignment strategies”, in Workshop on Mixed Criticality Systems (WMC), 2015, pp. 6-11.

Chapter 1

Introduction

1.1 Real-Time Cyber-Physical Systems

Cyber-Physical Systems (CPS) are systems that interact with the physical world through sensors and actuators, facilitate control and processing, and integrate with the cyber world of communication and computation [1]. A special class of CPS is employed in domains such as automotive, avionics and medical systems. These systems have stringent timing requirements, implying the need to deliver the output within the time interval imposed by the domain. Since a potential failure in these systems could cause catastrophic consequences to human lives and environment, the correctness of the system is verified in terms of functionality and timeliness. Such systems are called *real-time systems* [2]. The correctness of a real-time system depends not only on the logical correctness but also on the temporal correctness, *i.e.*, the computations must complete within their pre-specified timing constraints (referred as the *deadline*).

Based on the consequences of a deadline miss in the system, real-time systems are classified as: *hard* and *soft* real-time systems [3]. In hard real-time systems, *e.g.*, Flight Control System, a deadline miss can potentially lead to *catastrophic* consequences such as “death, system loss or severe environmental damage” [4]. Because of this reason, for systems (safety-critical) that have hard real-time requirements, there is a strong emphasis on verification and validation. On the other hand, a deadline miss in soft real-time systems, *e.g.*, Infotainment Systems, generally leads to a degraded quality of service.

1.1.1 Computation Platforms

The focus of this thesis is on the processor scheduling of real-time systems. Hence, in this section, we first describe the processing platform and later in Section 1.1.2 we describe the processor scheduling techniques.

Single-core platforms. A single-core hardware platform comprises a single processing core assumed to run at unit speed - x units of processing capacity in any time interval of length x . Nowadays, there are techniques (*e.g.*, Dynamic Voltage Frequency Scaling) to vary the processing speed at run time which may increase/reduce the processing capacity. Nevertheless, in this thesis, we consider only unit speed processing cores.

Multi-core platforms. Multi-core hardware platforms comprise m processing cores and each processing core is assumed to run at unit speed. A multi-core platform may either have identical processing cores (homogeneous) or distinct processing cores (heterogeneous). In this thesis, we only consider an identical multi-core platform executing at a unit speed called *homogenous unit capacity multi-core platform*. Resources such as memory and interconnect are not considered in our analysis. Henceforth, when we use the term resource we usually refer to processing cores, unless stated otherwise.

1.1.2 Real-Time Scheduling Theory

Workload models, scheduling policies and schedulability tests are often used to characterize and design a real-time system. The workload model defines the timing requirements such as deadlines and resource requirements of the system. Several abstract workload models [5–9], also called task models, have been designed in the past and all of them share the same principle: a function (piece of code) becomes available at some time instant and consumes a certain time duration for its execution, and is required to complete its execution within a time constraint.

Scheduling policies or in short ‘*schedulers*’ are the algorithms that make scheduling decisions such as when to execute a task on a processing core and for how long a task has to be executed. Scheduling policies are often evaluated for their capability to schedule a large number of task systems and for its runtime overhead and computational complexity. Schedulability test is a validation test usually done offline to check whether the given task system can be successfully scheduled (*deadline satisfaction*) by the scheduling algorithm.

The process of scheduling the task system on a single processing core is referred as single-

core scheduling. Multi-core scheduling refers to the process of scheduling the task system on a platform comprising of multiple processing cores. See [10] for an excellent survey on multi-core scheduling algorithms. Unlike single-core scheduling, multi-core scheduling is inherently a tedious process due to issues such as task mapping to cores, concurrency (parallelism) and interference from other tasks.

Based on how tasks are executed, multi-core scheduling can be classified into *global* and *partitioned* scheduling. Under global scheduling, tasks are allowed to execute on any processing core and can migrate between processing cores during runtime. Fluid scheduling, a type of global scheduling, allows fractional processing core allocation to tasks [11–15]. Thus, it is possible to simultaneously execute multiple tasks on the same processing core. However, fluid schedules cannot be realized on real hardware platforms and require transformation from fluid to non-fluid ones before it can be applied. Under partitioned scheduling, tasks are statically assigned to a processing core and not allowed to migrate during runtime.

1.2 Mixed-Criticality (MC) Systems

Most real-time systems found in automotive and avionics industries are designed to support a single isolated functionality on a dedicated processing core. However, in recent years, there is an increasing trend to reduce the Size, Weight and Power (SWaP) of the computing elements. This essentially promotes the integration of multiple functionalities on a common computing platform. This has led to the design of *mixed-criticality* systems. Contrarily, owing to the growing demand for functionalities in these systems, with the advent of multi-core platforms in the market, multi-core platforms seem an innate choice to co-host these applications. Integrated Modular Avionics (IMA) [16] and AUTomotive Open System ARchitecture (AUTOSAR) [17] are some industrial initiatives aimed at the integration of functionalities onto common hardware, while maintaining compliance with safety standards.

The concept of mixed-criticality can be interpreted in two ways. The straightforward one is that it supports multiple functionalities with different levels of ‘criticality’ (or *importance*) on a single hardware platform, where some functions (*e.g.* flight control) are more *critical* than others (*e.g.* cabin light control). The other interpretation is restricted to domains such as avionics and automotive that are subjected to mandatory certification requirements. Criticality in these domains refers to assurance level for a functionality against failures.

Criticality Level	Failure Condition
A	Catastrophic
B	Hazardous
C	Major
D	Minor
E	No effect

Figure 1.1: DO-178B classification of criticality levels

In avionics the criticality levels are usually termed as Design Assurance Levels (DAL), and in automotive they are termed as Automotive Safety and Integrity Levels (ASIL). A mixed-criticality system perhaps could have up to five criticality levels [18]. For example, standards such as DO-254, DO-178B and DO-178C support five criticality levels. The consequence of missing a deadline vary based on task criticality level. Figure 1.1 represents the safety levels and consequences according to the DO-178B standard. Deadline miss by a level A task might lead to catastrophic effect whereas, deadline miss by a level E task has no safety effect or might cause degraded service in the worst case [19].

The challenge in these systems is to manage the conflicting requirements of guaranteed execution of critical functionalities for safety assurance and efficient sharing of resources for improved quality of service. Hence, the fundamental question underlying MC systems is: *How to efficiently allocate resources while ensuring safe execution of the critical functionalities?*

1.2.1 Mixed-Criticality Scheduling

MC systems need to be certified before they are qualified for implementation. Certification authorities (*e.g.* Federal Aviation Administration, Nuclear Regulatory Commission) assess the safety standards of these systems, including the temporal correctness of the critical functionalities. These authorities tend to be very conservative in their assessment, such that the correctness of the system is verified under extremely rigorous and pessimistic assumptions. Certifications are aided by the analysis carried out on the abstract models of the system. The modeling process generally incorporates substantial pessimism in order for the analysis to hold for the eventually implemented systems.

The above modeling pessimism is unavoidable due to the uncertainties in the Worst-Case Execution Time (WCET) estimation of tasks. The WCET of a task is defined as the maximum possible time a task could take to execute on a specific hardware platform [20]. The execution time of a task may vary depending on the path taken by the control and the amount of time spent on the instructions in this path when executed on that hardware. Determining the exact WCET of a task has remained a challenging problem for years due to architectural features such as cache related delays, pre-fetching, shared interconnects, out-of-order execution, etc. The actual execution time of the task can never exceed the WCET value, otherwise, the temporal correctness of the system cannot be guaranteed by real-time scheduling theory. Thus, in practice, the WCET value used for analysis and scheduling is a conservative upper bound that exceeds the actual WCET [21].

Thus, a conservative WCET estimation of tasks ensures sufficient resource reservation for the critical functionalities. However, in practice, the probability that a task executes as much as this conservative WCET is very low. As a result, the resources provisioned for the critical functionalities are often left unused leading to under-utilization of system resources. Although such assumptions on WCET are required for safe system design, from the resource utilization perspective, they are very pessimistic. Hence, it is reasonable to use a less conservative execution budget that is sufficient for almost all execution scenarios that may occur in practice, as long as reasonable provisions exist to handle the unlikely scenarios. In addition, there would not be any serious consequence if any of the less critical tasks have an occasional deadline miss. Thus, it is not reasonable to require that the less critical tasks always meet their deadlines.

Based on the above motivation, Vestal [22] proposed a MC workload model that employs multiple WCET estimations and formally defined its scheduling requirements. The correctness of the functionalities is guaranteed at different criticality levels with their corresponding assurance level. In this model, the less critical tasks are guaranteed to meet their deadlines when all executions complete within their less conservative WCET estimates. If a task executes beyond its less conservative WCET estimate, a system level mode switch is triggered and thereafter, the critical tasks execute with their conservative WCET estimates. Later studies in MC scheduling make an assumption that the less critical tasks are no longer required to meet their deadlines once the system switches mode. The resource reserved for the less critical tasks could then be utilized by the critical tasks to guarantee correctness.

1.3 Motivation and Problem Statement

Since Vestal’s seminal work on MC workload model, several studies have been carried out in MC scheduling (see [18] for a comprehensive survey). Although scheduling of such systems on single-core platforms has been well explored, for multi-core platforms however, there has not been much work.

Most studies restrict their analysis to MC systems with two criticality levels - low-criticality and high-criticality. Such systems are known as dual-criticality MC systems where a task could either be a low-criticality task or a high-criticality task. In this thesis as well, we restrict our focus to such dual-criticality MC systems. For dual-criticality MC systems scheduled on homogeneous multi-core platforms, we address the following fundamental issues in this thesis.

1. **How can we improve the schedulability (defined in Chapter 2, Section 2.1.2, Definition 1) of state of the art global MC scheduling algorithm?** MC scheduling problem is known to be NP-Hard in the strong sense [23, 24]. As a result, the only known results are in the form of approximation algorithms that focus on improving the schedulability performance. Since optimal (in terms of schedulability) algorithms cannot be designed due to computational complexity, the effectiveness of these algorithms is often evaluated using theoretical performance bounds such as speed-up factor¹ (defined in Chapter 2, Section 2.1.3, Definition 3), and experimental evaluation. Scheduling algorithms with high schedulability in the experimental evaluation, as well as good theoretical performance bound (e.g., low speed-up factor), are generally preferred. However, most existing global MC scheduling algorithms suffer from poor schedulability [25–27]. To address this problem, a fluid execution rate based scheduling algorithm called *MC-Fluid* has been proposed for dual-criticality MC systems [28]. *MC-Fluid* is shown to outperform all existing global MC scheduling algorithms [28] and has an optimal speed-up factor of $4/3$ [29]. However, *MC-Fluid* is not capable of scheduling a large class of task systems, especially when their utilization (defined in Chapter 2, Section 2.1.1) is high. Thus, it is important to design a new scheduling algorithm that has better schedulability performance at high system utilization values and at the same time have an optimal speed-up factor. In the initial part of this thesis, we address this problem through a new fluid scheduling algorithm.

2. **Is there a better task-to-core mapping strategy that improves the schedulability per-**

¹Informally, the speed-up factor of an algorithm is defined as the maximum additional processing core speed required to schedule any feasible task system by that algorithm.

formance of partitioned MC scheduling?

In the latter part of this thesis, we focus on partitioned MC scheduling as they are preferred in industries and offers better schedulability compared to global variants. Partitioned scheduling comprises of task allocation to cores and scheduling of tasks on each core. The task-to-core mapping (partitioning/bin-packing) problem has been known to be NP-Complete [30]. Several partitioning heuristics have been designed in the past to assign tasks to cores. Existing studies [31–35] use different partitioning heuristics developed for classical real-time systems to assign MC tasks to cores. However, such conventional partitioning heuristics are very pessimistic for most MC schedulability tests. To address this problem, we propose a new task-to-core mapping strategy designed for MC systems.

3. **Since mode transition (formally described in Chapter 2, Section 2.2.2) is restricted to a particular core in partitioned MC scheduling, would it be possible to execute the low-criticality tasks with full service after mode transition by migrating them to another core?** Several studies in MC scheduling make an unrealistic assumption that all the low-criticality tasks can be penalized when the high-criticality tasks demand additional processing resource [25–28, 31–35]. Towards addressing this pessimism, in the final part of this thesis we focus on the support of low-criticality tasks. Several techniques have been proposed for single-core [36–54] and multi-core systems [55–58]. But, they do not utilize the benefit of partitioned MC scheduling. Under partitioned scheduling, the mode transition is restricted to an individual core due to isolation feature. As a result, upon mode transition, the low-criticality tasks executing on one core can be migrated to another, thus improving the service offered to them.

1.4 Contributions and Organization

The main contributions of this thesis are summarized below.

- Chapter 2 introduces the workload model used in real-time scheduling theory, scheduling algorithms and notations used in this thesis. It also presents the classical MC workload model and describes its execution semantics.
- Chapter 3 presents literature review related to the focus of this thesis: MC scheduling algorithms for single-core and multi-core platforms, and techniques to support the execution of low-criticality tasks.

- Chapter 4 presents our findings on improving the schedulability performance of global MC scheduling. Most existing global scheduling algorithms are very inefficient in scheduling a wide variety of dual-criticality MC systems. The state-of-the-art algorithm that focuses on improving schedulability is based on fluid scheduling and is also been shown to have an optimal performance bound in terms of the speed-up factor. Although it has an optimal speed-up factor, it suffers from poor schedulability especially when the system utilization is high. This is because it is very inefficient in managing the additional demand of the high-criticality tasks after mode transition. Thus, there is a need to design a new scheduling algorithm that has better schedulability performance than state-of-the-art and also retain optimal speed-up factor. In this chapter, we address the above shortcoming by proposing a new speed-up optimal fluid scheduling model for dual-criticality MC systems to efficiently handle the additional demand of the high-criticality tasks. We show that the proposed model outperforms the existing fluid model both theoretically and experimentally. In fact, we show that the schedulability test of the proposed model generalizes the test of the existing model. Our experiment results show that the proposed model is able to improve the schedulability by as much as 35.5% when compared to state-of-the-art.
- Chapter 5 studies the task-to-core mapping problem in the light of dual-criticality MC systems. Existing partitioned algorithms use conventional partitioning strategies designed for classical real-time systems. However, such partitioning schemes are very inefficient in balancing the additional demand of the high-criticality tasks after mode transition. In fact, the schedulability performance of several single-core MC scheduling algorithms depend on this additional demand. Hence, it is important to distribute this additional demand evenly among all processing cores. In this chapter, we address this issue by proposing a new utilization difference based partitioning strategy that balances this additional demand across all cores. Experimental evaluation using synthetic task sets suggest that such a mapping heuristic is necessary for good performance of partitioned MC scheduling. Our results show that the performance improvement using our strategies is as much as 52.9% and 33.1% when compared to existing algorithms with a known and without a known speed-up factor respectively. Our results also show that the performance of our algorithms improve significantly with increasing number of processing cores, indicating their scalability.
- Chapter 6 investigates the need for a dynamic partitioning model to support the execution of low-criticality tasks in the partitioned scheduling setting. Most existing studies penal-

ize all low-criticality tasks by either completely discarding them or executing them with reduced WCET estimates after mode transition. However, in practice, mode transitions seldom occur and it is very pessimistic to penalize all low-criticality tasks. Based on this motivation, a *semi-partitioned MC scheduling model* has been proposed for dual-criticality MC systems in which the low-criticality tasks are migrated at runtime from one core to another. In order to meet the deadlines of these migrating tasks, they are scheduled with reduced deadlines after mode transition. However, executing these tasks with such reduced deadlines impact the overall utilization of the processing core. To overcome this problem, in this chapter, we propose a reservation based approach to schedule these migrating tasks. Our experiment results show that using reservations improve the overall schedulability performance by as much as 24.3%, and offers as much as 29.2% additional low-criticality utilization compared to existing semi-partitioned MC algorithm.

- Chapter 7 summarizes this thesis and discusses some future directions for this work.

Chapter 2

Background

In this chapter, we introduce the classical workload model used in real-time scheduling theory and scheduling algorithms used in this thesis. We also present the classical mixed-criticality workload model and describe its execution semantics.

2.1 Real-Time System Descriptions

In this section, we first describe the classical workload model used in real-time scheduling theory, followed by scheduling algorithms and their performance metrics. We consider a discrete-time computing system where the events occur at distinct time points (derived from the system clock). All our parameters are defined based on this time unit.

2.1.1 Workload Model

Each instance of a functionality, called *job*, is a basic unit of computation and specified using the following parameters - the release time r , total resource requirement c and relative deadline d . The job is required to complete its c units of execution within the time interval $(r, r+d]$.

The workload model proposed in [6] is a typical recurrent task model found in many practical systems such as automotive and avionics. A recurrent task could be either periodic or sporadic. A periodic task τ_i is characterized by a triplet (T_i, C_i, D_i) , where T_i is the separation time between successive job releases (also called period), C_i is the resource requirement (usually the WCET of the task) and D_i denotes the relative deadline. Each task can potentially release an infinite

Figure 2.1: Periodic task $\tau_i = (10, 3, 8)$

number of job instances. Figure 2.1 shows a periodic task $\tau_i = (10, 3, 8)$ and its sample execution pattern. For sporadic tasks, T_i is the minimum inter-release separation time, *i.e.*, the separation time between two successive job releases. A task can also be classified based on its relative deadline. Based on the deadline, it is denoted as *implicit-deadline* if $D_i = T_i$, *constrained-deadline* if $D_i \leq T_i$, and *arbitrary-deadline* if $D_i \neq T_i$.

Utilization of a task is defined as the ratio of its resource requirement to its period. *System utilization* is defined as the total sum of the individual task utilizations. If a task set τ comprises of n tasks $\{\tau_1, \tau_2, \dots, \tau_n\}$, then the individual task utilization u_i and system utilization U is given as,

$$u_i = \frac{C_i}{T_i}, \text{ and}$$

$$U = \sum_{i=1}^n u_i.$$

2.1.2 Taxonomy of Real-Time Scheduling

Scheduling in real-time systems refers to the process of resource allocation to the tasks so as to meet its timing constraints (deadline). In order to schedule a task, its resource requirement, period and deadline need to be known. Scheduling policies or in short ‘*schedulers*’ are the algorithms that generate the schedule. It decides which job to be executed on which core at any time instant. For example, Rate Monotonic (RM) [7] scheduling executes the task with the smallest period (assigned the highest priority), and Earliest Deadline First (EDF) [7] scheduling first schedules the job with shortest remaining time to its absolute deadline ($r + d$). Schedulers are broadly classified into two based on the schedule generation time:

1. **Offline Scheduling:** Prior to runtime the schedule is generated. The exact release times and the deadlines of task instances must be known a priori.

2. Online Scheduling: Schedule is generated during runtime. Online schedulers, in turn, are classified into *fixed-priority* (e.g., RM) and *dynamic-priority* (e.g., EDF) schedulers.

- (a) Fixed-priority: Priorities are assigned at the task level, *i.e.*, all jobs of the same task have the same priority.
- (b) Dynamic-priority: Priorities are assigned at the job level. Different jobs of the same task may have a different priority depending on their release time, deadline or any other parameter.

Note that, the scheduling algorithms proposed in this thesis are all online algorithms.

Definition 1 (Schedulable) A task set τ is said to be schedulable under a scheduling algorithm $\mathbb{A}$ if for all possible release patterns every instance of a task $\tau_i \in \tau$ can execute C_i time units and signal completion within D_i time units from its release.

A schedulability test is used to validate the temporal correctness of the system at design time. A test is said to be *necessary* if all the schedulable task systems pass the test. A task system that passes the test may be schedulable, but not necessarily. A schedulability test is said to be *sufficient* if the task systems that pass the test are deemed schedulable. However, if the test fails, then it is not known whether the task system is schedulable or not. When a test provides both *necessary* and *sufficient* conditions, it is said to be *exact*. Although a schedulability test with low computational complexity is preferred, high-complexity tests are also useful if they improve the runtime performance, since these tests are mainly used for offline analysis.

2.1.3 Feasibility, Optimality and Speed-up Factor

A task system is said to be *feasible* on a platform if there exists a scheduling algorithm that can successfully schedule the task system on that platform. For homogenous multi-core platforms, the feasibility test for an implicit-deadline task system is given as follows.

Definition 2 (Property of feasible task systems [59]) An implicit-deadline task system τ is said to be feasible on a homogenous unit speed platform comprising m cores iff,

$$\sum_{\tau_i \in \tau} u_i \leq m$$

$$\forall i, u_i \leq 1$$

An *optimal* scheduling algorithm for the given platform can schedule all the feasible task systems. In the case of single-core scheduling, EDF is known to be an optimal scheduling algorithm [7]. Similarly, for multi-core platforms derivatives of fluid scheduling algorithms are known to be optimal [11–13].

When an optimal algorithm cannot be designed due to computational complexity, metrics such as speed-up factor are often used to quantify the performance of an algorithm [60]. The speed-up factor of a multi-core scheduling algorithm is formally captured in Definition 3.

Definition 3 (Speed-up Factor [33]) *A scheduling algorithm $\mathbb{A}$ is said to have a speed-up factor of ϕ ($\phi \geq 1$), if any feasible task system on m unit speed homogenous processing cores is successfully scheduled by $\mathbb{A}$ on $m\phi$ speed homogenous processing cores.*

The speed-up factor is used to characterize the effectiveness of a scheduling algorithm. A scheduling algorithm with a speed-up factor of ϕ is said to be *speed-up optimal* if no other algorithm can have a better (smaller) speed-up than ϕ . In general, algorithms with smaller speed-up factor (close to 1) are preferred over algorithms with a larger speed-up factor.

2.1.4 Fluid Scheduling

In this thesis, we consider the fluid scheduling policy for MC systems. Hence, in this section, we formally define the fluid scheduling policy and present some important results from literature.

Fluid scheduling is a type of global scheduling that allows fractional processing core allocation to tasks. Under fluid scheduling, each task fairly progresses at every time unit such that the task temporal constraints are met. Figure 2.2 shows the execution of a task τ_i under fluid scheduling.

Figure 2.2: Fluid execution of a task τ_i

Fluid scheduling platform is defined as the platform in which, at all time instants, tasks are

executed on a hypothetical fractional (< 1) processing core. Thus, it is possible to simultaneously execute multiple tasks on the same core. The rate at which the tasks are executed by the fluid scheduling platform is defined as the execution rate.

Definition 4 (Execution rate [28]) *A task τ_i is said to be executed with execution rate $\theta_i \in \mathbb{R}^+$, s.t. $0 < \theta_i \leq 1$, if every job of the task is executed on a fractional processing core having a speed of θ_i .*

To guarantee the schedulability of the task system under fluid scheduling, the following conditions need to be satisfied.

1. Platform feasibility: The total execution rates of all the tasks in the system do not exceed the total processing capacity of the computing platform.
2. Task Schedulability: The execution rate of each task is sufficient to complete its execution by its deadline.

The following definitions present the platform feasibility and task schedulability conditions for fluid scheduling on a homogenous multi-core platform.

Definition 5 (Platform feasibility [11]) *The execution rates assigned to task system τ are feasible on a homogenous multi-core platform comprising m unit speed cores iff,*

$$\sum_{\tau_i \in \tau} \theta_i \leq m.$$

Definition 6 (Task schedulability [11]) *The implicit-deadline task system τ is schedulable iff,*

$$\forall \tau_i \in \tau, \quad \theta_i \cdot T_i \geq C_i.$$

For classical real-time task systems scheduled on multi-core platforms, several derivatives of fluid scheduling algorithms are known to be optimal for implicit-deadline task systems [11–13, 15]. However, fluid schedules cannot be realized on an actual hardware platform. Hence, to realize the benefit of fluid schedules, there have been several studies focused on converting fluid schedules to non-fluid schedules with minimal or no loss in performance [14, 61]. The drawback of these approaches is that they allow a large number of preemptions which results in high overhead costs.

2.2 MC System Descriptions

In this section, we present the MC workload model proposed by Vestal and describe its execution semantics [22].

MC systems are subjected to the certification process whose objectives are usually complementary to those of the system designers. System designers are responsible for ensuring the correctness of the entire system whereas, certification authorities are concerned only about the correctness of the critical parts of the system. From the certification perspective, the less critical applications are occasionally allowed to fail, as they do not impact the safety of the system, when the system is overloaded due to the critical applications. Based on this trait of MC systems, Vestal proposed a new workload model to abstract MC system and defined its scheduling requirements [22]. The key aspect of Vestal's model is that the correctness of the system is guaranteed at different levels based on the execution mode. During normal mode of operation all tasks are required to meet their deadlines, and when the critical tasks demand additional resources, *i.e.*, overloaded mode, the less critical tasks are no longer required to meet their deadlines.

2.2.1 MC Workload Model

The workload model proposed by Vestal has been widely used in MC research. The model extends the classical real-time workload model (presented in Section 2.1.1) to the MC setting. Each MC task is associated with a criticality level that denotes the required level of assurance against failure. Let $\mathfrak{L} \in \mathbb{Z}^+$ denote the number of criticality levels in the system. Then, each task is assigned a criticality level $\chi_i \in \{1, 2, \dots, \mathfrak{L}\}$, and can have multiple WCET estimations that vary based on criticality levels. The MC workload model can be formally defined as follows.

Definition 7 (MC Workload [22]) *Each MC task τ_i is characterized by a tuple (T_i, χ_i, C_i, D_i) , where*

- $T_i \in \mathbb{Z}^+$ is the period or minimum release separation time,
- $\chi_i \in \mathbb{Z}^+$ is the criticality level and $\chi_i = 1$ denotes the lowest criticality level.
- $C_i \in \{C_i^1, C_i^2, \dots, C_i^{\chi_i}\} \in \mathbb{R}^+$ is a vector of WCET estimations - one for each criticality level; we assume $C_i^1 \leq C_i^2 \leq \dots \leq C_i^{\chi_i}$. This is reasonable because as the criticality level increases the required confidence level in these WCET estimations also increases.

- $D_i \in \mathbb{Z}^+$ is the relative deadline.

Most existing studies on MC scheduling have been done on dual-criticality MC systems. In a generic dual-criticality MC system, there are tasks with one of two criticality levels, namely low-criticality (LC) task and high-criticality (HC) task. That is, $\chi_i \in \{LC, HC\}$ and $C_i \in \{C_i^L, C_i^H\}$. Note that, C_i^H is defined only for HC tasks and there is no C_i^H for LC tasks.

The LC task set τ_L is defined as $\tau_L \stackrel{\text{def}}{=} \{\tau_i \in \tau \mid \chi_i = LC\}$ and the HC task set τ_H is defined as $\tau_H \stackrel{\text{def}}{=} \{\tau_i \in \tau \mid \chi_i = HC\}$. The utilization of a MC task in such a system is defined as follows.

Definition 8 (MC task utilization) *The LC and HC utilization of a MC task τ_i are defined as*

$$u_i^L \stackrel{\text{def}}{=} \frac{C_i^L}{T_i}, \text{ and} \quad (2.1a)$$

$$u_i^H \stackrel{\text{def}}{=} \frac{C_i^H}{T_i}. \quad (2.1b)$$

Then, the total utilization of a MC task system at different criticality levels is defined as follows.

Definition 9 (MC system utilization) *The LC system utilization of LC tasks (U_L^L) and HC tasks (U_H^L) are defined as the total LC utilization of LC tasks and HC tasks respectively. The HC system utilization of HC tasks (U_H^H) is defined as the total HC utilization of HC tasks.*

$$U_L^L \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_L} u_i^L \quad (2.2a)$$

$$U_H^L \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_H} u_i^L \quad (2.2b)$$

$$U_H^H \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau_H} u_i^H \quad (2.2c)$$

2.2.2 Execution Semantics

In this section, we describe the execution semantics of the MC workload model for a dual-criticality MC system. The system is said to be either in the LC mode or in the HC mode.

The system initially starts in the LC mode. If all tasks signal completion before exceeding their LC WCET estimation, the system remains in the LC mode. In this mode all task deadlines

are required to be met. If any HC task executes beyond its LC WCET estimation, but does not signal completion, the system switches to the HC mode. The system remains in this mode until it can be safely returned back to the LC mode². No LC task deadlines are required to be met after the mode switch from LC to HC, and hence the system could (if so required) immediately discard all the LC tasks. This is reasonable to assume because the system has to prioritize all the HC tasks over LC tasks so that there are sufficient resources reserved for the more conservative WCET estimation of the HC tasks. However, in practice, LC tasks can be executed as background workload. Figure 2.3 depicts these execution modes of a dual-criticality MC system.

Figure 2.3: MC execution modes

Definition 10 (Mode switch instant) *Mode switch instant is defined as the first time instant in a busy interval when the system mode changes from LC to HC.*

Since the correctness of a dual-criticality MC system depends on the correctness of the tasks in different execution modes, such a system is defined to be *MC-schedulable* by a scheduling policy if it satisfies the following two conditions.

Definition 11 (MC-Schedulable) *A dual-criticality task set τ is said to be MC-schedulable by a scheduling algorithm if,*

- *LC guarantee: Each task in τ is able to complete execution up to its LC WCET estimation within its deadline in LC mode, and*
- *HC guarantee: Each task in τ_H is able to complete execution up to its HC WCET estimation within its deadline in HC mode.*

It can be observed that the WCET estimation for a criticality level is being used as a budget for task τ_i at that criticality level. At runtime, it is used by the scheduler to determine when

²This could happen either at an idle instant in the schedule or earlier if some accounting technique to ensure safety is employed.

to switch mode and prioritize HC tasks over LC tasks. Therefore, henceforth we use the term *budget* to denote the WCET estimations.

Mode switch in multi-core MC scheduling. There is a fundamental difference in the behaviors of global versus partitioned MC scheduling algorithms when a mode switch is triggered as shown in Figure 2.4. Under global MC scheduling when any of the HC tasks on a core triggers a mode switch, it is reflected instantaneously on the other cores, meaning all LC tasks could have an impact thereafter. Whereas, in the case of partitioned MC scheduling, the mode switch is restricted only to that particular core. Essentially, the tasks executing on the other cores, including LC tasks, are unaffected by the mode switch. This isolation between cores is feasible under partitioned scheduling because the MC schedulability tests are applied independently on each core. It is an important property because it also reduces the impact on LC tasks, and this could be one of the reasons why partitioned MC scheduling may find preference in safety-critical industries.

Figure 2.4: Mode switch in multi-core MC scheduling

2.3 MC Scheduling Algorithms

In this section, we describe the various MC scheduling policies considered in this thesis. First, we describe the Earliest Deadline First with Virtual Deadlines (EDF-VD) algorithm designed for single-core MC systems. Although the EDF-VD algorithm is designed for single-core MC

systems, it can be directly applied to multi-core MC systems through partitioned MC scheduling. Second, we describe the dual-rate fluid scheduling model designed for global scheduling of multi-core MC systems.

2.3.1 Earliest Deadline First with Virtual Deadlines (EDF-VD) Algorithm

The EDF scheduling policy schedules the job with the shortest absolute deadline irrespective of its criticality level. Although the EDF algorithm is efficient for classical real-time systems, for MC systems it is very inefficient. This is mainly because EDF is unable to handle the sudden increase in the demand of HC tasks upon a mode switch.

Under EDF-VD, each HC task is assigned a virtual deadline (D_i^L) in the LC mode based on a single scaling factor $x \in (0, 1)$, *i.e.*, $D_i^L = x \cdot T_i$ [62]. By assigning a shorter deadline in the LC mode, jobs of HC tasks are given priority over LC jobs with shorter absolute deadlines. This enables the scheduler to better handle the sudden increase in demand of HC jobs upon a mode switch. These virtual deadlines are determined by uniformly distributing the LC mode slack among all the HC tasks.

In the LC mode, LC tasks are scheduled using their actual deadlines and HC tasks are scheduled using their virtual deadlines. Upon a mode switch, all the LC tasks are dropped and thereafter the HC tasks are scheduled using their actual deadlines. For implicit-deadline dual-criticality MC systems, an utilization based schedulability test for EDF-VD with an optimal speed-up factor of 4/3 has been developed. This schedulability condition is given as follows.

Definition 12 (EDF-VD Schedulability [62]) *An implicit-deadline dual-criticality MC task system τ is said to be schedulable by the EDF-VD algorithm on a unit speed single-core platform if,*

$$U_L^L \leq \frac{1 - U_H^H}{1 - (U_H^H - U_H^L)} \quad (2.3)$$

Similar to EDF-VD, there are other studies [63–65] based on assigning virtual deadlines to HC tasks in the LC mode, and use EDF for scheduling. But, unlike EDF-VD, these studies use a demand bound function³ based test and a greedy deadline assignment strategy. The main challenge in these studies is to bound the execution demand of the carry-over HC jobs, *i.e.*, jobs

³The demand bound function gives an upper bound on the maximum possible execution of a task system in any given time interval.

whose execution demand is increased after the mode switch.

Definition 13 (Carry-Over Job) *A job of HC task τ_i is said to be a carry-over job, if it is released before mode switch instant but has not completed its execution until mode switch.*

Although these studies improve the schedulability performance over EDF-VD, they do not have a known speed-up bound.

2.3.2 Dual-rate Fluid Scheduling Algorithm

Multi-core MC scheduling algorithms are in general complex and inefficient in terms of schedulability compared to single-core algorithms. In order to address this issue, Lee *et al.* [28] proposed a dual-rate fluid scheduling model, called MC-Fluid, for implicit-deadline dual-criticality MC systems scheduled on homogenous multi-core platforms. The proposed algorithm is known to have the best schedulability performance among all the existing global algorithms.

Under the fluid scheduling model, it is possible to allocate fractional processing capacities (< 1) to tasks, as long as the total processing capacity allocated to all the tasks at each time instant does not exceed the total capacity of the platform. By allowing fractional processing capacities, the schedulability of the algorithm is greatly improved.

The dual-rate fluid scheduling model was designed to schedule implicit-deadline (*i.e.*, deadline equal to period) dual-criticality MC systems. The model derives its name from the fact that each HC task is assigned two execution rates; one for the LC mode execution and another for the HC mode execution. On the other hand, each LC task is assigned a single execution rate to be used in the LC mode.

Definition 14 (Dual-rate model [28]) *The dual-rate fluid scheduling model can be summarized as follows.*

- Each task $\tau_i \in \tau$ executes at a constant rate θ_i^L (where $\theta_i^L \in [u_i^L, 1]$) in the LC mode,
- All tasks in τ_L are discarded immediately upon mode switch, and
- Each task $\tau_i \in \tau_H$ executes at a constant rate θ_i^H (where $\theta_i^H \in [u_i^H, 1]$) in the HC mode.

For the dual-rate model, a sufficient schedulability test has also been derived as follows.

Figure 2.5: Mode switch in dual-rate fluid scheduling

Definition 15 (Dual-rate Schedulability, Theorem 1 in [28]) *An implicit-deadline dual-criticality task system τ is said to be MC-schedulable under dual-rate fluid scheduling on a homogenous multi-core platform comprising m unit speed cores iff,*

$$\forall \tau_i \in \tau, \quad \theta_i^L \geq u_i^L \quad (2.4)$$

$$\forall \tau_i \in \tau_H, \quad \frac{u_i^L}{\theta_i^L} + \frac{u_i^H - u_i^L}{\theta_i^H} \leq 1 \quad (2.5)$$

$$\forall \tau_i \in \tau_H, \quad \theta_i^H \geq \theta_i^L \quad (2.6)$$

$$\sum_{\tau_i \in \tau} \theta_i^L \leq m \quad (2.7)$$

$$\sum_{\tau_i \in \tau_H} \theta_i^H \leq m \quad (2.8)$$

Equations (2.7) and (2.8) give the platform feasibility conditions. This ensures that the execution rates in each mode are feasible on the multi-core platform comprising m cores. The task schedulability condition in the LC mode is given by equation (2.4). This ensures each task is able to receive budget proportional to its utilization within its deadline. Similarly, the task schedulability condition in the HC mode, including that of the carry-over job, is given by equations (2.5) and (2.6). The HC mode schedulability test is derived based on the worst-case mode switch instant of an HC task. It has been shown in [28] that the worst-case mode switch instant occurs when a carry-over job completes C_i^L units of execution at mode switch as shown in Figure 2.5. In this case, the mode switch is triggered by the task itself. Intuitively, the carry-over job is left with the shortest possible duration in the HC mode to execute its remaining demand $C_i^H - C_i^L$ with the higher rate θ_i^H . Since equation (2.6) is not necessary, the definition itself only presents

a sufficient test. With the help of an example we now illustrate the schedulability of dual-rate model.

Example 1 Table 2.1 shows an MC system scheduled on a dual-core platform under the dual-rate fluid scheduling, where the task set τ and corresponding rate assignments are given. We need to show that Eqs. (2.4)-(2.8) hold for the given rate assignments so that the system is MC-Schedulable. We can see that Eqs. (2.4) and (2.6) hold. We now show that Eq. (2.5) holds: $\{0.2/0.571 + (0.85 - 0.2)/1\} \leq 1$ for τ_1 , $\{0.25/0.472 + (0.5 - 0.25)/0.531\} \leq 1$ for τ_2 , $\{0.15/0.283 + (0.3 - 0.15)/0.319\} \leq 1$ for τ_3 , and $\{0.1/0.15 + (0.15 - 0.1)/0.15\} \leq 1$ for τ_4 . We can check that $\sum_{\tau_i \in \tau} \theta_i^L = \{0.571 + 0.472 + 0.283 + 0.15 + 0.2\} \leq 2$ (Eq. (2.7)) and $\sum_{\tau_i \in \tau_H} \theta_i^H = \{1 + 0.531 + 0.319 + 0.15\} \leq 2$ (Eq. (2.8)).

Table 2.1: Example implicit-deadline task system and its rate assignment.

	T_i	C_i^L	C_i^H	χ_i	u_i^L	u_i^H	θ_i^L	θ_i^H
τ_1	10	2	8.5	HC	0.2	0.85	0.571	1
τ_2	20	5	10	HC	0.25	0.5	0.472	0.531
τ_3	30	4.5	9	HC	0.15	0.3	0.283	0.319
τ_4	40	4	6	HC	0.1	0.15	0.15	0.15
τ_5	50	10	10	LC	0.2	-	0.2	-
$\sum$					0.9	1.8	1.676	2.000

Execution Rate Assignment. An optimal execution rate assignment (OERA) strategy has also been proposed [28]; that is, OERA is guaranteed to find a feasible dual-rate assignment for a dual-criticality MC system if one exists. OERA determines the values of the HC mode execution rates by solving a convex optimization problem (Definition 16). The LC mode execution rates (Definition 17) are then determined in polynomial time.

Definition 16 (θ_i^H assignment) Given an implicit-deadline dual-criticality MC system τ , the HC mode execution rates $\theta_i^H (\forall \tau_i \in \tau_H)$ is the solution to the following optimization problem.

$$\begin{aligned}
 & \text{minimize} && \sum_{\tau_i \in \tau_H} \frac{u_i^L(u_i^H - u_i^L)}{\theta_i^H - u_i^H + u_i^L} \\
 & \text{subject to} && \sum_{\tau_i \in \tau_H} \theta_i^H \leq m
 \end{aligned} \tag{2.9a}$$

$$\forall \tau_i \in \tau_H, \quad \theta_i^H \geq u_i^H \tag{2.9b}$$

$$\forall \tau_i \in \tau_H, \quad \theta_i^H \leq 1 \tag{2.9c}$$

Definition 17 (θ_i^L assignment) Given an implicit-deadline dual-criticality MC system τ and the θ_i^H rates for each HC task $\tau_i \in \tau_H$, the LC mode execution rates θ_i^L are given by,

$$\theta_i^L = \begin{cases} \frac{u_i^L \cdot \theta_i^H}{\theta_i^H - u_i^H + u_i^L}, & \text{if } \tau_i \in \tau_H, \\ u_i^L, & \text{otherwise.} \end{cases} \quad (2.10)$$

MC-Derivative Rate Assignment. In [66], we propose another optimal execution rate assignment strategy called MC-Derivative that has a lower computational complexity than OERA and better scalability for multi-criticality systems. To determine the execution rates, OERA relies on a rigid convex optimization framework. To overcome this dependence, we proposed MC-Derivative that uses first-derivative principles to determine the execution rates.

The intuition behind MC-Derivative is as follows. Let us recall the objective of the rate assignment problem defined in Definition 16.

$$f(\theta_i^H) = \sum_{\tau_i \in \tau_H} \frac{u_i^L(u_i^H - u_i^L)}{\theta_i^H - u_i^H + u_i^L}$$

Let θ_i^H be defined as $u_i^H + X_i$. Then, the objective function can be rewritten as a function of X_i given by, $f(\vec{X}) = \sum f_i(X_i)$, where $\vec{X}$ denotes $\{X_i\}$ and $f_i(X_i)$ denotes $\frac{u_i^L(u_i^H - u_i^L)}{X_i + u_i^L}$. The aim of MC-Derivative is to minimize $f(\vec{X})$. Since $f(\vec{X})$ is differentiable and convex, the derivative technique by first principle can be applied to determine the optimal solution.

The derivative by first principle, $f'_i(X_i) = \frac{df(\vec{X})}{dX_i}$, denotes the change in function $f_i(X_i)$ with respect to a change in X_i . Since the derivative $f'_i(X_i)$ is a negative function in the range of X_i (i.e., $u_i^H \leq \theta_i^H \leq 1$), we represent the magnitude of $f'_i(X_i)$ using $\bar{f}'_i(X_i)$. Thus, a higher $\bar{f}'_i(X_i)$ represents a larger decrease in $f(\theta_i^H)$. MC-Derivative minimizes $f(\theta_i^H)$ using this observation. All tasks are initialized with $X_i = 0$, the least X_i which makes $f(\theta_i^H)$ to be the largest. Then, it increases X_i of the task with largest $\bar{f}'_i(X_i)$. The procedure is repeated until there is no slack (defined as $m - \sum \theta_i^H$) left in the system, thus minimizing $f(\theta_i^H)$. Below, we illustrate the MC-Derivative assignment strategy with an example.

Example 2 Consider the example task set presented in Table 2.1. Fig. 2.6 plots the corresponding $\bar{f}'_i(X_i)$ functions for varying X_i . MC-Derivative initially assigns $X_i = 0$, the least feasible execution rate, for all tasks. Then, it increases X_i of task τ_1 while its $\bar{f}'_i(X_i)$ is the largest among all. The value of X_1 is increased until it becomes the maximum value, i.e., $1 - u_1^H$

Figure 2.6: The $\bar{f}'_i(X_i = x)$ plot for varying x

or the slack is zero. For the example case, $X_1 = 1 - u_1^H$. Next, τ_2 and τ_3 are chosen as they have the second largest $\bar{f}'_i(X_i)$. So, X_2 and X_3 are increased simultaneously while their $\bar{f}'_i(X_i)$ are the largest. The slack becomes 0 before X_4 could be increased. At this point, **MC-Derivative** ends, and τ_4 is left with an initial assignment of $X_4 = 0$. Note that, the black dotted line Γ in Fig. 2.6 denotes the objective value corresponding to zero slack. We can see that, $\bar{f}'_1(1 - u_1^H) > \bar{f}'_2(0) = \bar{f}'_3(0) > \Gamma > \bar{f}'_4(0)$.

Thus, **MC-Derivative** assigns $X_1 = 1 - u_1^H$, X_i of task τ_2 and τ_3 between 0 and $1 - u_i^H$ and $X_4 = 0$. The LC mode execution rates are assigned by Definition 17. Then, the system is said to be MC-Schedulable if it satisfies Equation (2.7).

MC-Derivative Correctness. The MC-Derivative algorithm assigns HC mode execution rates (θ_i^H) for HC tasks. The θ_i^H assignment problem in Definition 16 is constructed based on the dual-rate schedulability conditions. Hence, the MC-Derivative algorithm is correct if it satisfies Equations (2.9a)-(2.9c). It can be seen that MC-Derivative assigns rates until slack ($m - \sum \theta_i^H$) becomes zero. Thus, Equation (2.9a) is satisfied. The MC-Derivative initially assigns $X_i = 0$ for all tasks i.e., $\theta_i^H = u_i^H$, and thereby, satisfying Equation (2.9b). Also, for any task it assigns a maximum rate of $1 - u_i^H$. As a result, Equation (2.9c) is satisfied.

MC-Derivative Scalability. Next, with the help of an example we illustrate the scalability of

MC-Derivative to implicit-deadline MC systems with three criticality levels.

Table 2.2: Example 3-Criticality implicit-deadline task system scheduled on dual-core platforms

	T_i	C_i^L	C_i^M	C_i^H	χ_i	u_i^L	u_i^M	u_i^H
τ_1	10	1	5	6	HC	0.1	0.5	0.6
τ_2	10	3	4	6	HC	0.3	0.4	0.6
τ_3	10	1	3	5	HC	0.1	0.3	0.5
τ_4	10	2	4	-	MeC	0.2	0.4	-
τ_5	10	1.5	-	-	LC	0.15	-	-

Example 3 Consider an example 3-criticality (low-criticality – LC, medium-criticality – MeC and high-criticality – HC) task system as shown in Table 2.2. The execution semantics of the system is as follows:

- The system initially starts in the LC mode
- If any MeC task or HC task executes more than its C_i^L execution budget, the system switches to the MC mode
- If any HC task executes more than its C_i^M (denoted as MeC budget) execution budget, the system switches to the HC mode

Let θ_i^L , θ_i^M and θ_i^H denote the execution rates in the LC, MeC and HC mode respectively. Then, the OERA algorithm can be applied to the 3-criticality MC system if we decompose the rate assignment problem into sub-problems as follows:

1. Minimize $\sum_{\tau_H} \theta_i^M$ subject to $\sum_{\tau_H} \theta_i^H \leq m$.
2. Minimize $\sum_{\tau_M \cup \tau_H} \theta_i^L$ subject to $\sum_{\tau_M \cup \tau_H} \theta_i^M \leq m$.

where τ_M denotes the MeC task set. We can determine the $\{\theta_i^H\}$ rates by applying the OERA algorithm to the first subproblem and subsequently determine the $\{\theta_i^M\}$ rates by solving the second subproblem. The resulting execution rates for the example task system is unschedulable (i.e., $\sum \theta_i^L > 2$) as shown in Table 2.3.

We show that the extension of MC-Derivative can find a schedulable rate assignment for the example task system. Let the objective function $f(\vec{X})$ be defined as $\sum_{\tau_M \cup \tau_H} \theta_i^L$. Then, MC-Derivative uses derivative by first principles to determine the execution rates, i.e., differentiate

Table 2.3: Rate assignment to the example 3-criticality task system

	OERA			MC-Derivative		
	θ_i^L	θ_i^M	θ_i^H	θ_i^L	θ_i^M	θ_i^H
τ_1	0.5274	0.6214	0.6000	0.3911	0.6925	0.6000
τ_2	0.5493	0.5493	0.7359	0.6277	0.4000	0.7351
τ_3	0.4293	0.4293	0.6641	0.3278	0.5075	0.6649
τ_4	0.4000	0.4000		0.4000	0.4000	
τ_5	0.1500			0.1500		
$\sum$	2.0560	2.000	2.000	1.8966	2.000	2.000

$f(\vec{X})$ with respect to θ_i^M and θ_i^H :

$$f_i''(X_i) = \frac{\partial^2 f(X)}{\partial \theta_i^M \partial \theta_i^H}.$$

The **MC-Derivative** initially assigns the minimum valid rate of $\theta_i^M = u_i^M$ and $\theta_i^H = u_i^H$ for all MeC tasks and HC tasks. Then, it increases the θ_i^M rate of the tasks in the order of their maximum $f_i''(X_i)$ derivative value as long as the MeC mode slack ($m - \sum_{\tau_M \cup \tau_H} \theta_i^M$) is non-negative. Similarly, it increases the θ_i^H rate to the HC tasks in the order of their maximum $f_i''(X_i)$ derivative value as long as the HC mode slack ($m - \sum_{\tau_H} \theta_i^H$) is non-negative. As shown in Table 2.3, the resulting rate assignment for the same example task system is schedulable ($\sum \theta_i^L \leq 2$, $\sum_{\tau_M \cup \tau_H} \theta_i^M \leq 2$, and $\sum_{\tau_H} \theta_i^H \leq 2$).

Thus, the **MC-Derivative** has better flexibility for extension to multi-criticality systems compared to the **OERA** algorithm.

Chapter 3

Related Work

The mixed-criticality (MC) scheduling model was first introduced by Vestal in 2007 [22]. Since then, MC scheduling has gained a lot of attention; see [18] for a complete review on MC systems and scheduling. Baruah [23] showed that the complexity of MC scheduling problem is NP-hard in the strong sense by a reduction from the 3 – *partition* problem [30]. As a consequence, only approximation algorithms have been designed for MC systems. In this chapter, we present past research studies in MC systems related to the objectives of this thesis.

3.1 Single-core MC Scheduling

Single-core MC scheduling can be classified into *job scheduling* and *task scheduling* depending on whether the scheduling problem is considered for a set of jobs or tasks.

Job scheduling is a very constrained scheduling model where the scheduling problem is restricted to a finite set of MC jobs with mode dependent execution budgets. Initial studies on MC systems are based on job scheduling [23, 67, 68]. Later studies on MC systems consider a more general task model where each task could potentially release an infinite number of job sequences.

Most works on MC scheduling address the problem of scheduling a set of periodic or sporadic MC tasks. Typically, task scheduling can be categorized into fixed-priority and dynamic-priority scheduling.

Fixed-Priority Scheduling. Fixed-priority MC scheduling utilizes response time analysis (RTA) to determine schedulability. RTA is used to determine the worst-case response (completion) time of a task. A task system is said to be schedulable if the response time of all tasks

is within their respective deadlines. The very first work to apply RTA to MC systems is by Vestal [22]. The priority values of the tasks are assigned using Audsley's algorithm [69] assuming all tasks can execute at all the criticality levels. Thus, it is possible for LC tasks to have higher priorities than HC tasks even in the HC mode. This pessimistic assumption results in low system utilization.

To address the above shortcoming, Baruah *et al.* [70] presented an algorithm called Static Mixed Criticality (SMC) for MC systems with arbitrary criticality levels. Under SMC, in each mode, the jobs are not allowed to execute beyond their execution budgets for that mode. SMC and other prior work on fixed-priority MC scheduling do not drop LC tasks upon a mode switch. Baruah *et al.* [71] presented a fixed-priority algorithm called Adaptive Mixed Criticality (AMC) for constrained-deadline dual-criticality MC systems and derived its RTA based schedulability test. Under AMC, each task has a fixed-priority and all LC tasks are immediately dropped upon a mode switch. Since a worst-case mode switch instant is unknown, the AMC test considers all possible mode switch instants until the LC mode response time of the task. Burns and Davis [72] also incorporated deferred preemption in AMC, where each task has a non-preemptive region at its execution tail.

Dynamic-Priority Scheduling. Baruah *et al.* [73] proposed a virtual deadline based EDF algorithm called EDF-VD for dual-criticality implicit-deadline task systems, where each HC task is assigned a virtual deadline to be used in the LC mode that is no larger than the actual deadline. These virtual deadlines are assigned using a single scaling factor so that the slack in the LC mode is uniformly distributed among all HC tasks. Tasks are then scheduled using EDF in both modes, and LC tasks are immediately dropped upon a mode switch. A utilization-based schedulability test has also been developed. Later, Baruah *et al.* [62] improved the test with a speed-up factor of $4/3$, which is the optimal speed-up factor for any MC scheduling algorithm. Masrur *et al.* [74] extended the analysis to include two deadline scaling factors for HC tasks - one for a small increase in workload and another for a large increase in workload.

Ekberg and Yi [64] introduced a demand bound function based schedulability test for constrained deadline dual-criticality systems by considering the demand of LC and HC tasks separately. Under this scheme, HC tasks are assigned a virtual deadline in LC mode and EDF is used for scheduling. They also introduced a deadline tightening strategy for HC tasks called *Greedy* and showed that their algorithm performs better than EDF-VD. Easwaran [63] improved the demand bound function test of Ekberg and Yi by considering the demand bound function of both LC and HC tasks together and showed its dominance over the existing approach. A deadline

tightening strategy for HC tasks called Earliest Carry-over Deadline First (ECDF) has also been proposed. Demand bound function based tests are known to perform well in practice and can be applied to constrained-deadline task systems, but they do not have a known speed-up bound.

3.2 Multi-core MC Scheduling

As the recent trend in the chip industry is towards multi-cores, there have been some studies on multi-core MC scheduling as well in recent years ([26–29, 33–35, 57, 75, 76]). Most of the multi-core algorithms are designed based on either global or partitioned scheduling.

3.2.1 Global Scheduling

In global scheduling, tasks are allowed to execute on any of the processing cores and can migrate between cores during runtime. Initial work on multi-core MC scheduling is by Anderson *et al.* [75] based on two-level hierarchical scheduling in which they proposed a mix of partitioned and global approaches for MC systems with five criticality levels. They proposed an architecture to support the execution of different criticality tasks on different cores. Later, Herman *et al.* [77] presented the challenges and issues in implementing the architecture proposed in [75] on a real-time operating system and discussed some design tradeoffs to reduce scheduler overheads.

Pathan *et al.* [27] proposed a global fixed-priority scheduling algorithm based on RTA for constrained-deadline MC systems. A sufficient schedulability test has also been derived for the proposed algorithm. The proposed test is used to determine the priority ordering of the tasks based on Audsley's approach. An important feature of the proposed test is that it can be easily adapted for a general MC system with multiple criticality levels. Li and Baruah [26] proposed a global scheduling algorithm combining the multi-core fixed-priority algorithm fpEDF and single-core virtual deadline based MC algorithm EDF-VD for implicit-deadline dual-criticality MC systems, and proved that the algorithm has a speed-up factor of $\sqrt{5} + 1$.

Lee *et al.* [28] presented a fluid scheduling model for implicit-deadline dual-criticality MC systems called MC-Fluid, in which each task is assigned two execution rates (dual-rate) depending on the execution mode. Fluid scheduling allows tasks to be allocated fractional processing capacity, which, although practically infeasible, significantly improves the schedulability performance. They also proposed an optimal rate assignment algorithm for such systems and showed that MC-Fluid has a speed-up factor of $(\sqrt{5} + 1)/2$. Baruah *et al.* [29] proposed a simpli-

fied execution rate assignment algorithm for the dual-rate model called MCF and proved that both MC-Fluid and MCF are speed-up optimal with a speed-up factor of $4/3$. Ramanathan and Easwaran [78] presented different rate assignment strategies that have better performance than MCF and has lower computational complexity than MC-Fluid. Recently, Lee *et al.* [66] presented a technique to convert a MC-Fluid schedule to a non-fluid schedule with minimal loss in performance. *Although these algorithms are known to be dual-rate and/or speed-up optimal, there are several feasible dual-criticality task systems that are deemed not schedulable by them.*

Most existing global algorithms are inefficient in terms of scheduling a large class of MC systems. The problem of efficient multi-core MC scheduling has largely remained open. To address this issue, the dual-rate fluid scheduling model was developed. However, even the optimal dual-rate model MC-Fluid is not capable of scheduling a large number of task systems, especially when their system utilization is high. In Chapter 4, we address this issue by proposing a speed-up optimal multi-rate fluid scheduling model where each HC task executes with more than two rates. Immediately upon entering the HC mode, several HC tasks may simultaneously require additional processing capacity. The dual-rate model is very inefficient in managing this overload, and thus, when the system utilization increases, the performance of MC-Fluid decreases rapidly. To efficiently manage this overload, the multi-rate model distributes the execution across several windows of fixed duration. By managing the execution rates in these windows, the multi-rate model is able to provide a higher average execution rate to each HC task and thereby improving the schedulability performance. The experimental evaluation using synthetic task sets show that the proposed multi-rate fluid scheduling model improves the schedulability by as much as 35.5% when compared to the dual-rate model. We also show that the proposed multi-rate model generalizes the dual-rate model and thus, have an optimal speed-up factor of $4/3$ for implicit-deadline dual-criticality MC systems.

3.2.2 Partitioned Scheduling

Partitioned MC scheduling is motivated by the fact that 1) they have a better schedulability performance compared to global MC scheduling, and 2) safety-critical industries have a preference for them because they are a natural extension of single-core scheduling. The partitioned MC scheduling problem comprises two main challenges: 1) Statically assigning tasks to cores (*partitioning strategy*), and 2) Scheduling the tasks on each core using single-core MC scheduling algorithms.

The partitioning strategy can be classified into two types: *criticality-aware* partitioning in which tasks of a higher criticality are assigned to cores before tasks of a lower criticality, and *criticality-unaware* partitioning in which no such allocation order exists. The choice of partitioning strategy has a significant impact on the performance of any partitioned scheduling algorithm. Although for classical real-time systems, first-fit decreasing utilization scheme is known to be the best performing strategy, for MC systems there is no known partitioning strategy that performs well in all cases.

There have been few studies on partitioned MC scheduling algorithms. Kelly *et al.* [31] introduced fixed-priority, partitioning-based multi-core MC scheduling for implicit-deadline MC systems and showed that decreasing criticality and first-fit partitioning performs well in comparison to decreasing utilization and worst-fit partitioning. Baruah *et al.* [33] combined the EDF-VD schedulability test with a simple first-fit partitioning strategy to derive a partitioned scheduling algorithm with a speed-up bound of $8/3$ for implicit-deadline dual-criticality MC systems. In fact, this work showed that any partitioning strategy which considers all cores for allocation of a task before declaring failure has the same speed-up bound of $8/3$ when used with this test (Theorem 9 in [33]). Through experimental evaluation, they also showed that the partitioned algorithm performs better than the corresponding global EDF-VD algorithm in terms of schedulability. Gratia *et al.* [79] extended the single-core server based scheduling algorithm RUN, developed for classical real-time systems, to implicit-deadline dual-criticality periodic MC systems. The HC and LC tasks are initially allocated to servers and the servers are then assigned to cores for execution.

Rodriguez *et al.* [34] evaluated different partitioning strategies under EDF-VD and showed that criticality-aware partitioning performs better than criticality-unaware partitioning for dual-criticality MC systems. Gu *et al.* [35] presented a criticality-aware partitioning strategy for dual-criticality MC systems with worst-fit allocation for HC tasks and first-fit allocation for LC tasks, additionally also giving preference to heavy utilization LC tasks. They applied this partitioning strategy to a demand bound function based test and virtual deadline based algorithm presented in [64]. Han *et al.* [80] presented a criticality-aware partitioning strategy where tasks are assigned to a core with worst-fit allocation based on ‘utilization contribution’, defined as the ratio of task utilization to system utilization at respective criticality levels.

Existing partitioned scheduling algorithms are based on conventional partitioning heuristics (*e.g.*, sorting based on task utilization) developed for classical real-time systems. These strategies are highly inefficient in mapping MC tasks to cores, due to unbalanced load across cores when

the cores switch mode. The assignment of tasks to cores has a significant impact on the overall performance of a partitioned MC scheduling algorithm. It could either favor the scheduling algorithm applied on each core or adversely affect it. Furthermore, there has not been a partitioning heuristic that performs well across a wide variety of single-core MC scheduling algorithms. Towards addressing these issues, we propose a utilization difference based partitioning strategy for dual-criticality MC systems in Chapter 5. This strategy evenly distributes the additional demand of the HC tasks after mode transition (a key characteristic of a dual-criticality MC system) among all processing cores to minimize the pessimism in the single-core schedulability test applied on each core. We propose two variants of this strategy; criticality-aware and criticality-unaware partitioning. Our experiment results show that the schedulability improvement using our strategies is as much as 52.9% with EDF-VD when compared to the strategy proposed in [33]. Our proposed strategy when combined with the EDF-VD test also has a speed-up bound of $8/3$ for implicit-deadline dual-criticality MC systems. Our results also show that the proposed strategies, when combined with the AMC scheduling algorithm, perform well compared to existing dynamic-priority algorithms [34, 35]. The AMC scheduling algorithm for single-core MC systems is known to perform poorly when compared to dynamic-priority algorithms. From our results, it can be observed that the performance improvement is due to the partitioning strategies, rather than the difference in schedulability tests.

3.2.3 Semi-Partitioned Scheduling

Semi-partitioned scheduling is an extension of partitioned scheduling in which a subset of the tasks is allowed to migrate between cores. Semi-partitioned scheduling was developed with the intent of improving the schedulability performance while maintaining the advantages of partitioning. There have been numerous studies on semi-partitioned scheduling algorithms in the real-time systems domain. On the contrary, there are very few studies for MC systems.

Burns *et al.* [81] proposed a semi-partitioned scheduling of cyclic executives for dual-criticality MC systems. Cyclic executives, a restricted form of static scheduling, consists of a series of frames executing continuously. The frames are executed in sequence and the execution is repeated once the series is completed. The study constructed the schedule only for a collection of MC jobs. Awan *et al.* [82] extended different semi-partitioned algorithms designed for classical real-time systems to constrained-deadline dual-criticality MC systems. In this study, the LC tasks are allowed to migrate between cores for better utilization, while the HC tasks are statically

assigned to a core for safe execution.

3.3 Improving the Execution of LC Tasks

Most existing studies on MC scheduling employ a very pessimistic approach that completely discards/ignores all the LC tasks upon mode transition to guarantee resources for the HC tasks. This seriously impacts the performance which may not be suitable for many practical systems that require minimum service guarantees for these tasks [37, 50]. To overcome this problem, several techniques have been proposed in the past for single-core [36–54] and multi-core MC systems [55–58]. These approaches can broadly be classified as follows.

1. *Elastic Scheduling*. Under elastic scheduling, service is offered to the LC tasks by reducing their dispatch frequency (extending their periods) in the HC mode [37, 43, 50–52, 58, 83].
2. *Imprecise Computation/Reduced execution*. The LC tasks are executed with reduced execution budget when the system is in the HC mode [36, 37, 45, 46, 54, 56].
3. *Selective Degradation*. Depending on the budget available in the HC mode, only a certain subset of LC jobs/tasks are executed [39, 40, 44, 47, 48, 53, 57, 84].
4. *Processor speedup*. Huang *et al.* [42, 85] proposed a dynamic processor speedup technique to guarantee resources for HC tasks instead of degrading the service to the LC tasks in the HC mode.
5. *Delayed Penalty*. To delay dropping of the LC tasks, the mode switch instant is postponed as much as possible [49, 54, 84].
6. *Timely recovery*. Santy *et al.* [86] and Bate *et al.* [87] proposed some techniques to minimize the duration for which the system is in the HC mode. By doing so, they were able to reduce the limited-service duration of LC tasks.

3.3.1 Multi-core Techniques

Most multi-core MC scheduling algorithms, including the ones presented earlier, are based on the impractical assumption that all the HC tasks simultaneously require their conservative execution budgets upon a mode switch. As a result, all the LC tasks in the system are penalized

for providing resources to the HC tasks. All the partitioned MC scheduling algorithms that restrict the mode switch to a core by design, continue to schedule LC tasks executing on the other cores. However, LC tasks executing on the core that switches to the HC mode are ignored. To address this issue, techniques developed for the support of LC tasks in single-core MC systems can be used. There have been a few such studies on service guarantee schemes for LC tasks under partitioned scheduling.

Elastic Scheduling. Su *et al.* [58] extended the single-core elastic task model [50] for multi-core dual-criticality MC systems using partitioned EDF scheduling. Under this model, the LC tasks are released early depending on the slack generated from the over-provisioned HC task execution budgets.

Imprecise Computation. Pathan [56] extended the imprecise MC model [36] with the quality of service values for LC tasks and presented a fluid execution based scheduling algorithm for implicit-deadline dual-criticality MC systems. Under this model, based on the slack available in the HC mode, the LC tasks are executed with their actual LC execution budgets instead of reduced budgets.

Selective Degradation. Ren *et al.* [57] presented a task grouping scheme, that can isolate different task groups by restricting the impact of mode switch to within that group, for implicit-deadline dual-criticality MC systems. Under this technique, each HC task is grouped with some LC tasks and only these tasks are affected if that particular HC task causes a mode switch. The LC tasks executing in different task groups are unaffected by it. Zaid *et al.* [76] introduced a dual-partitioned scheduling model for implicit-deadline dual-criticality MC systems. In their model, the HC tasks are statically mapped to a core, and the LC tasks in a core are migrated upon mode switch under the assumption that all the cores switch mode simultaneously. Their model assumes a fixed-partition for each of the criticality modes, and hence the name dual-partitioned scheduling. However, their model did not consider schedulability during the mode switch transition, and hence is unsafe. Further, the LC tasks active during the mode switch are dropped and these tasks execute at a lower frequency in the destination core after the mode switch.

Delayed Penalty. Xu *et al.* [88] considered a semi-partitioned model for constrained-deadline dual-criticality MC systems based on RTA [71], which allows the LC tasks to continue execution upon mode switch by completely migrating them to the other core which has not experienced the mode switch. However, the proposed model is restricted to dual-core platforms. This model

guarantees full service to the LC tasks when any one of the cores switches mode. If both cores switch mode, then all the LC tasks in the core that switched mode last are dropped to meet the demand of the HC tasks on both these cores. Later, Xu [89] extended this work to platforms with four cores and explored different migration model for the LC tasks where these tasks are allowed to migrate more than once. All the LC tasks in the system are executed with their actual execution budgets as long as the number of cores that switch mode is within the pre-specified limit.

Most existing multi-core MC studies [56–58, 76] on service guarantee schemes for LC tasks do not consider full service to all the LC tasks (execution with their actual budget) after the mode switch. In practice, the probability that a HC task overruns its budget is very low and degrading the service of the LC tasks if one such event occurs is very pessimistic. Hence, it is better to offer full service to all the LC tasks instead of being dropped or executed with a reduced budget upon mode switch. In partitioned MC scheduling, this can be achieved by migrating the LC tasks upon mode switch to another core. Based on this motivation, Xu *et al.* [88] proposed the semi-partitioned MC model using AMC scheduling algorithm for dual-criticality MC systems. Under this model, in order for the migrating LC tasks to be schedulable in the core to which it migrates, these tasks are scheduled with the reduced deadline after the mode switch. However, scheduling these tasks with such reduced deadlines is very pessimistic as it significantly affects the overall utilization of cores. To overcome this limitation, in Chapter 6, we propose a reservation-based scheduling model for implicit-deadline dual-criticality MC systems to guarantee full service for all the LC tasks when one of the cores switches to the HC mode. Under this model, the migrating LC tasks are scheduled using servers and the utilization corresponding to these tasks are reserved in the core to which it migrates. By using reservations we were able to improve the overall utilization of each core. Our experiment results show that the proposed scheduling algorithm improves the overall schedulability performance by as much as 24.3% compared to existing algorithm [88], and offers as much as 29.2% additional LC utilization compared to [88]. We do not compare the performance of our algorithm with the one presented in [89] because this work supports all LC tasks as long as the number of cores that switch mode is within a pre-specified bound and allows LC tasks to migrate to more than one processing core.

There are several studies on multi-core MC systems that deal with parallel tasks, energy-aware scheduling, shared resource access, bounding contention delay, memory interference, etc. However, all these studies are orthogonal to the focus of this thesis, and hence, we do not present them here.

Chapter 4

Generalized and Efficient MC Fluid Scheduling Model

In this chapter, we present a new fluid scheduling model, called the *multi-rate model*, for implicit-deadline dual-criticality MC systems scheduled on homogeneous multi-core platforms. Immediately upon entering the HC mode in dual-criticality MC systems, several HC tasks demand additional processing resource. To efficiently manage this high variability in demand after mode transition, the multi-rate model executes the HC tasks with varying execution rates after the mode switch. By efficiently managing this demand variability, the multi-rate model is able to provide a better schedulability. This model has been previously published in [90].

We present the limitations of the existing global MC scheduling algorithms and present the motivation for this work in Section 4.1. In Section 4.2, we briefly describe the assumed system and present the limitations of existing dual-rate fluid scheduling model. We illustrate motivation for this work in Section 4.3. Section 4.4 introduces the multi-rate fluid scheduling model. In Section 4.5, we derive the sufficient schedulability test for the multi-rate model and show that it dominates the dual-rate model schedulability test in Section 4.6. Section 4.7 presents the technique to convert fluid schedules to non-fluid ones. Section 4.8 presents the rate and window duration assignment strategy for the multi-rate model, and prove that it is speed-up optimal. In Section 4.9, we evaluate the multi-rate model with existing algorithms and show that it outperforms all the other multi-core MC scheduling algorithms with known speed-up factors. In Section 4.10, we discuss the efficacy of the multi-rate fluid scheduler with respect to optimality and practical implementation. We conclude this chapter in Section 4.11.

4.1 Introduction

In Chapter 3, Section 3.2.1, we discussed several global MC algorithms [26, 28, 29, 66, 78] designed for scheduling dual-criticality MC systems on homogenous multi-core platforms. However, most of these algorithms have poor schedulability performance as discussed in Section 1.3. To evaluate the schedulability performance of these algorithms two techniques are generally used: 1) *experimental evaluation* in which the schedulability is assessed over a wide variety of task systems, and 2) analytical performance bounds such as the speed-up factor (defined in Chapter 2, Section 2.1.3, Definition 3) are derived. Scheduling algorithms with high schedulability in experimental evaluation as well as good (low) speed-up factor are highly desirable.

Among the algorithms presented in Section 3.2.1, fluid scheduling algorithms have demonstrated good performance in the experimental evaluation, and at the same time have bounded performance guarantee for task systems that are hard to schedule. The dual-rate fluid scheduling model called MC-Fluid [28], is known to have the best schedulability performance among all existing global MC scheduling algorithms (refer to Chapter 2, Section 2.3.2 for more details on the dual-rate model). Although algorithms such as MC-Fluid are dual-rate and speed-up optimal, there are several feasible dual-criticality MC systems that are not schedulable under the dual-rate model. In fact, in experimental evaluations, it has been observed that the performance of MC-Fluid drops significantly for task systems with utilization greater than $3/4$ [78]. The primary reason for this underperformance is the fact that a single fixed execution rate for each HC task after the mode switch is not efficient to handle the high variance in demand during the transition period immediately following the mode switch.

To address the aforementioned limitation of the dual-rate model, in this chapter, we propose a *multi-rate fluid scheduling model* for dual-criticality MC systems scheduled on homogenous multi-core platforms. Under this model, each HC task executes using a single execution rate in the LC mode and a set of execution rates in the HC mode. In particular, the task executes using different rates over time until all carry-over jobs complete their execution, and the subsequent HC jobs released in the HC mode execute using a single rate based on task utilization. Similar to the dual-rate model, each LC task uses a single execution rate before mode switch and is suspended after the mode switch. Thus, by using a fine-grained rate allocation in the transition period, the multi-rate model is able to accommodate a higher demand after the mode switch. This in turn enables it to accommodate a higher demand in the LC mode, thus improving schedulability over the dual-rate model.

4.2 System Description

In this chapter, we assume that workload of the system is represented using the implicit-deadline sporadic task system described in Section 2.2.1. We focus on the global scheduling of dual-criticality systems, and the execution semantics of the system and scheduling criteria is presented in Section 2.2.2. Since no LC task deadlines, including for jobs that are released before the mode switch, are required to be met in HC mode, it is possible to drop all the LC task jobs immediately upon mode switch. The HC task jobs in HC mode can be classified into two types depending on their release time - *carry-over job* and *pure HC mode jobs*. As defined in Chapter 2, Section 2.3.1, Definition 13, a job of task τ_i is said to be a carry-over job, if it is released before mode switch instant but has not completed its execution until mode switch. All the remaining HC mode jobs of τ_i , those that are released after the mode switch, are called pure HC mode jobs. The carry-over and pure HC mode jobs of all the HC tasks must be guaranteed HC execution budgets by their deadlines. Similarly, the pure LC mode jobs of all the tasks, those with deadlines before mode switch, must be guaranteed LC execution budgets by their deadlines.

4.2.1 Dual-rate fluid scheduling model and its limitation

We consider the dual-rate fluid scheduling model, MC-Fluid [28], presented in Section 2.3.2. The dual-rate model was designed to schedule implicit-deadline dual-criticality MC systems on homogeneous multi-core platforms. It assigns different execution rates to tasks in each criticality level. Each HC task executes using two rates and each LC task executes using a single rate. The execution semantics of the dual-rate MC-Fluid model is presented in Chapter 2, Section 2.3.2, Definition 14. The HC task rates depend on the operating mode of the system; they execute using higher rate once the system switches to the HC mode, while the LC tasks are suspended upon mode switch. The schedulability analysis of MC-Fluid is presented in Chapter 2, Section 2.3.2, Definition 15.

Although the dual-rate model improves the schedulability of state-of-the-art global MC scheduling algorithms, they are inefficient at high system utilization ranges as discussed in Section 1.4. The prime reason being their capability to handle the varying demand of carry-over jobs upon mode switch. The demand of the carry-over job after the mode switch is usually higher than the pure HC mode jobs. The total demand of HC tasks in the HC mode vary depending on the number of such carry-over jobs. This overall demand affects the total demand that can be ac-

commodated in the LC mode, and in turn affecting the schedulability performance. The use of single execution rate in the HC mode for the HC tasks is an inefficient way to handle this varying demand. Thus, there is a need to manage the carry-over jobs demand in the transition period after the mode switch more productively. In the next section, we illustrate with an example to handle this varying demand with multiple execution rates.

4.3 Motivational Example

The difference between the multi-rate and other models is how they handle carry-over jobs after the mode switch. There are two types of jobs for each HC task; a single carry-over job which in the worst-case has an execution requirement of $C_i^H - C_i^L$ between the mode switch instant and its deadline, and pure HC mode jobs which have an execution requirement of C_i^H between their release time and deadline. So in the transition period after mode switch while such carry-over jobs of different HC tasks are active, there is an incentive to use different execution rates.

In the multi-rate model, the job execution rates change depending on the number of active carry-over jobs at any time instant. Once all the carry-over jobs complete, each HC task use a single execution rate, which is equal to the task utilization. By doing this fine-grained rate adjustment in the transition period as opposed to the existing algorithms, multi-rate model is able to accommodate a higher workload for the carry-over jobs after mode switch. This in turn reduces the workload of HC tasks in the LC mode, thus allowing it to accommodate more LC task workload.

Now we use an example to explain the difference between various models in terms of how they handle the execution of HC tasks after mode switch. In particular, we present the results of using:

1. A single execution rate after mode switch (as in the dual-rate model),
2. Two execution rates after mode switch (slight modification of the dual-rate model in which we can use one rate for the carry-over job and another for the pure HC mode jobs), and
3. Multiple execution rates after mode switch (multi-rate fluid model).

Example 4 *In Table 4.1 we show a dual-criticality task system that is schedulable under the multi-rate model on 2 core, but is not schedulable under the dual-rate model.*

Tasks	C_i^L	C_i^H	T_i	u_i^L	u_i^H	Dual-rate assignment		Multi-rate assignment							
						θ_i^L	θ_i^H	θ_i^L	$\theta_{i,1}^H$	w_1	$\theta_{i,2}^H$	w_2	$\theta_{i,3}^H$	w_3	$\theta_i^H (= u_i^H)$
τ_1	2.8	4.9	7	0.4	0.7	0.700	0.700	0.571428	1.000	2.1	0.7	0.4	0.7	13.76	0.7
τ_2	1.5	4	5	0.3	0.8	0.641	0.939	0.600	1.000	2.1	1.000	0.4	0.8	13.76	0.8
τ_3	3.5	10.5	35	0.1	0.3	0.224	0.360	0.186766	0.000	2.1	0.300	0.4	0.500	13.76	0.3
τ_4	15.75	-	35	0.45	-	0.450	-	0.450	-	-	-	-	-	-	-
$\sum$						2.015	1.999	1.808294	2.000		2.000		2.000		1.80

Table 4.1: Example task system schedulable under the multi-rate fluid model, but not under the dual-rate fluid model on dual-core platforms. C_i^L denotes LC execution budget, C_i^H denotes HC execution budget, T_i denotes task period, $u_i^L = C_i^L/T_i$ and $u_i^H = C_i^H/T_i$. For the dual-rate fluid model, θ_i^L denotes execution rate in the LC mode and θ_i^H denotes execution rate in the HC mode. For the multi-rate fluid model, θ_i^L denotes execution rate in the LC mode, $\theta_{i,j}^H (j \in \{1, 2, 3\})$ and θ_i^H denote several execution rates for the HC mode, and $w_j (j \in \{1, 2, 3\})$ denotes duration of time for which rate $\theta_{i,j}^H$ will be used by task τ_i . After $w_1 + w_2 + w_3$ time units in the HC mode, each HC task τ_i will use the rate θ_i^H .

Under the dual-rate model, both the carry-over job and the pure HC jobs of a HC task execute using a single rate. As can be seen, using the dual-rate model, the resulting execution rates in LC mode are not feasible ($\sum_{\tau_i \in \tau} \theta_i^L = 2.015 > 2$). Since these rates have been obtained using the dual-rate optimal **MC-Fluid** algorithm, we can conclude that the task system is not schedulable under the dual-rate model.

Let us suppose a slight modification of the dual-rate model in which the carry-over job uses the execution rate determined by **MC-Fluid** and the pure HC jobs use another execution rate based on task utilization does any better than the dual-rate model. Using different execution rates for the carry-over and pure HC mode jobs does not help in reducing the LC mode rates any further than the dual-rate model. This is because the execution rate of the carry-over job determines the rate allocation of that task for the LC mode.

In the case of multi-rate model, each HC task is assigned multiple execution rates in the HC mode as shown in the table. Immediately after entering the HC mode in window w_1 , while tasks τ_1 and τ_2 execute at the maximum rate of 1, τ_3 does not execute at all. Later on in window w_3 , when τ_1 and τ_2 are executing at a much lower rate equal to their utilization, τ_3 is able to execute at a rate of 0.5. This is even higher than the rate it was allocated in the dual-rate model (0.36). Thus, the average rate allocated to τ_3 across windows w_1, w_2 and w_3 in the multi-rate model is higher than the rate allocated to it in the dual-rate model. As a result its rate in LC mode is reduced ($\sum_{\tau_i \in \tau_H} \theta_i^L = 1.808 \leq 2$), and the overall task system becomes schedulable (schedulability is formally verified in Section 4.5). The schedule of this task set in the high-

Figure 4.1: Schedule in the high-criticality mode

criticality mode is shown in Figure 4.1 for both dual-rate and multi-rate models.

Thus, it is possible to schedule task systems that are not schedulable by dual-rate fluid model by assigning multiple rates for tasks executing after the mode switch in the system. In the next section, we formally define the multi-rate model and its execution semantics.

4.4 Multi-rate Fluid Scheduling Model

Under the multi-rate fluid scheduling model, each task $\tau_i \in \tau$ executes with a single rate in the LC mode as in the dual-rate model. But, unlike the dual-rate model, each task $\tau_i \in \tau_H$ executes with at most $n_H + 1$ rates in the HC mode, where $n_H = |\tau_H|$ is the number of HC

tasks. As discussed in Section 4.3, the intuition behind these multiple rates in the HC mode can be explained as follows. When a mode switch occurs, there are potentially n_H carry over jobs in the system, all of which may require an execution rate higher than their HC utilization (i.e., u_i^H). By allowing them to adjust their execution rates several times in the *transition period* immediately after a mode switch, it may be possible to accommodate more carry over demand in the system.

Definition 18 (Multi-rate Fluid Model) *The multi-rate fluid model can be formally defined as follows:*

- *LC mode execution rate:* Each job of task $\tau_i \in \tau$ will start executing at a rate θ_i^L in the LC mode. If there is no mode switch while the job is active, then this is the only rate at which it will execute.
- *Transition execution rates:* Upon a mode switch, all the LC tasks will be immediately dropped. The transition period after the mode switch is partitioned into n_H transition windows ($j : 1 \leq j \leq n_H$), each of a fixed duration w_j . The jobs of a HC task $\tau_i \in \tau_H$ execute at a constant rate of $\theta_{i,j}^H$ in each transition window j .
- *HC mode execution rate:* After the completion of the transition period (after $\sum_j w_j$ time units from the mode switch), each job of HC task $\tau_i \in \tau_H$ executes at a constant rate of θ_i^H .

Thus, we denote the bounded period of time between the mode switch and $\sum_j w_j$ time units thereafter as the *transition period*. Note that the execution rates in the transition period can be used either by carry-over jobs or pure HC mode jobs that are released in the transition period. We denote the HC mode jobs released in the transition period as **transition jobs** and the HC mode jobs released after the transition period as **stable jobs**. It is also worth noting that in this multi-rate model the n_H transition window durations are all determined offline, and remain fixed at runtime. As a consequence, similar to the dual-rate model, the runtime scheduling mechanism for this multi-rate model is also very simple.

The proposed multi-rate model is a generalization of the dual-rate model that was discussed in Section 2.3.2. If we set all the transition execution rates to be equal to θ_i^H , then in the resulting model, each task executes with a fixed rate in LC mode and a fixed rate in HC mode. This setting is identical to the dual-rate model. We will now proceed to derive the schedulability conditions of the multi-rate model.

4.5 Schedulability Test

In the previous section, we presented the multi-rate model for dual-criticality MC system. In Section 2.3.2, we presented the dual-rate schedulability test and the worst-case mode switch pattern for which the execution rates need to be determined. In this section, we derive a sufficient schedulability test for the multi-rate fluid scheduling model defined in Section 4.4.

For any fluid scheduling algorithm, the schedulability analysis comprises of task schedulability and platform feasibility as mentioned in Section 2.1.4. In the case of fluid scheduling for dual-criticality MC systems, this derivation is composed of four steps:

1. **LC mode task schedulability test:** We derive a test using LC mode execution rates to ensure that each task is schedulable in the LC mode.
2. **LC mode platform feasibility test:** We derive a test using LC mode execution rates and the number of cores to ensure that the allocated LC mode rates are feasible on the homogeneous multi-core platform.
3. **HC mode platform feasibility test:** We derive a test using HC mode execution rates and the number of cores to ensure that the allocated HC mode rates are feasible on the homogeneous multi-core platform. Here, we need to consider all the $n_H + 1$ rates assigned to each HC task.
4. **HC mode task schedulability test:** We derive several conditions using LC mode and HC mode execution rates to ensure that each HC task is schedulable in the HC mode. We need to consider carry-over jobs, transition jobs as well as stable jobs in this step.

4.5.1 LC mode task schedulability test

In the LC mode, jobs of each task $\tau_i \in \tau$ execute with a single rate θ_i^L . It is easy to see that $\theta_i^L \geq u_i^L$ is a necessary condition for task schedulability in LC mode. Since our framework is based on fluid scheduling, this is also a sufficient schedulability condition in the LC mode. If the task receives an execution rate of at least u_i^L , then the total allocated budget to each job of the task is at least $u_i^L \times T_i$, which is sufficient to meet deadlines. We record this test in the following proposition.

Proposition 1 *The implicit-deadline task system τ is schedulable in the LC mode iff*

$$\forall \tau_i \in \tau, \theta_i^L \geq u_i^L.$$

4.5.2 LC mode platform feasibility test

In the LC mode, a set of execution rates is feasible on a homogeneous multi-core platform comprising m cores if and only if the sum total of the rates is no more than m . If the total is more than m , then clearly the rates cannot be assigned using m cores. Whereas, if it is no more than m , then it can be assigned because a single core can schedule multiple tasks at the same time under the fluid scheduling model.

Proposition 2 *The execution rates assigned to task system τ are feasible in the LC mode iff*

$$\sum_{\tau_i \in \tau} \theta_i^L \leq m.$$

4.5.3 HC mode platform feasibility test

Similar to the LC mode case, feasibility of the assigned execution rates is ensured as long as the sum total of the rates is no more than m . However, since we have multiple execution rates in the HC mode, we need to ensure that rates are feasible in each of the $n_H + 1$ windows.

Proposition 3 *The execution rates assigned to task system τ are feasible in the HC mode iff*

$$\forall j(1 \leq j \leq n_H), \sum_{\tau_i \in \tau_H} \theta_{i,j}^H \leq m, \text{ and} \quad (4.1)$$

$$\sum_{\tau_i \in \tau_H} \theta_i^H \leq m. \quad (4.2)$$

4.5.4 HC mode task schedulability test

In this section, we derive sufficient schedulability conditions for tasks in the HC mode. There are three types of jobs to consider; carry-over jobs that are released before the mode switch but remain unfinished in the LC mode, transition jobs that are released in the transition period (within $\sum_{j:1 \leq j \leq n_H} w_j$ time units from the mode switch), and stable jobs that are released after the transition period. We first consider the stable jobs and then derive the tests for the other jobs.

4.5.4.1 Schedulability test for stable HC mode jobs

In this section, we derive the schedulability conditions for stable jobs in the HC mode. Since these jobs are released after the transition period, they always execute using a single rate θ_i^H . As long as $\theta_i^H \geq u_i^H$ these jobs can meet their deadlines because $u_i^H * T_i = C_i^H$ units of execution would be guaranteed. It is easy to see that this is also a necessary condition for schedulability of the stable jobs; if $\theta_i^H < u_i^H$ then the stable jobs cannot meet their deadlines. We record this test in the following proposition.

Proposition 4 *A stable job of task τ_i , that is a job released at or after $\sum_{j:1 \leq j \leq n_H} w_j$ time units from the mode switch, is schedulable iff*

$$\theta_i^H \geq u_i^H$$

4.5.4.2 Schedulability tests for carry-over jobs

Figure 4.2: Worst-case mode switch instant of a carry-over job

In this section, we derive the schedulability conditions for carry-over jobs in the HC mode. Let k_i denote the largest index ($1 \leq k_i \leq n_H + 1$) such that the earliest deadline of any carry-over job of task τ_i is strictly greater than $\sum_{j:1 \leq j < k_i} w_j$ time units after the mode switch (see Figure 4.2). That is, if a carry-over job of τ_i executes for exactly C_i^L time units and triggers the mode switch, then it has the earliest possible deadline in the HC mode ($T_i - C_i^L / \theta_i^L$ time units from the mode switch instant). In this case, its deadline would either fall in the k_i^{th} transition window if $k_i \leq n_H$ or after the transition period if $k_i = n_H + 1$. Formally, k_i can be defined as follows.

Figure 4.3: A carry-over job that does not trigger a mode switch

Definition 19 (Earliest completion window k_i)

$$\sum_{j:1 \leq j < k_i} w_j < T_i - C_i^L/\theta_i^L,$$

and if $k_i \leq n_H$, then

$$\sum_{j:1 \leq j \leq k_i} w_j \geq T_i - C_i^L/\theta_i^L.$$

Note that no job of task τ_i (carry-over or otherwise) can have a deadline within $\sum_{j:1 \leq j < k_i} w_j$ time units of the mode switch instant. This is an important property of k_i that we will use in the following derivations. The below theorem derives a sufficient schedulability test for the carry-over jobs. Similar to the dual-rate model, the main intuition behind this theorem is that the worst case scenario for a carry-over job occurs when the job itself triggers the mode switch. It means that the job has executed with the smaller rate θ_i^L for the longest possible duration in the LC mode and left with the shortest possible duration (i.e., $T_i - C_i^L/\theta_i^L$) in the HC mode to complete its remaining execution (i.e., $C_i^H - C_i^L$).

Theorem 1 A carry-over job of task $\tau_i \in \tau_H$ can meet its deadline in the HC mode if the following three conditions are met.

$$\sum_{j:1 \leq j < k_i} \theta_{i,j}^H \times w_j + R_i \times \left(T_i - C_i^L/\theta_i^L - \sum_{j:1 \leq j < k_i} w_j \right) \geq C_i^H - C_i^L \quad (4.3)$$

$$\forall j : k_i \leq j \leq n_H, \theta_i^L \leq \theta_{i,j}^H \quad (4.4)$$

$$\theta_i^L \leq \theta_i^H \quad (4.5)$$

where $R_i = \theta_{i,k_i}^H$ if $k_i \leq n_H$ and $R_i = \theta_i^H$ otherwise.

Proof: **Only if:** We first show that Equation (4.3) is a necessary condition for the schedu-

lability of a carry-over job that triggers the mode switch. But since Equations (4.4) and (4.5) are not necessary, the theorem itself only presents a sufficient test. If Equation (4.3) does not hold then this carry-over job will miss its deadline. Consider the carry-over job shown in Figure 4.2. Since it triggers the mode switch, its remaining execution time at mode switch is $C_i^H - C_i^L$, and the time remaining to deadline is $T_i - C_i^L / \theta_i^L$. From the definition of k_i we get $\sum_{j:1 \leq j < k_i} w_j < T_i - C_i^L / \theta_i^L$, and the total execution for the carry-over job in this window is $\sum_{j:1 \leq j < k_i} \theta_{i,j}^H * w_j$. Thus, the remaining execution of $C_i^H - C_i^L - \sum_{j:1 \leq j < k_i} \theta_{i,j}^H * w_j$ must be provided in the remaining time window of $T_i - C_i^L / \theta_i^L - \sum_{j:1 \leq j < k_i} w_j$. This is only possible if Equation (4.3) holds.

If: Assuming Equation (4.3) holds, the total execution a carry-over job triggering a mode switch receives after the mode switch is $\geq C_i^H - C_i^L$. Since it has already received C_i^L units of execution before mode switch, it can meet its deadline. Let us consider the case of a carry-over job that does not trigger the mode switch as shown in Figure 4.3. In this case, the carry-over job has a deadline greater than $T_i - C_i^L / \theta_i^L (> \sum_{j:1 \leq j < k_i} w_j)$. Suppose its deadline is at some time instant t . Then, from Equations (4.4) and (4.5) we get that each execution rate assigned to this carry-over job in the time interval $[T_i - C_i^L / \theta_i^L, t]$ is at least as much as θ_i^L . Hence, the amount of execution over and above $C_i^H - C_i^L$ remaining at mode switch $((t - T_i - C_i^L / \theta_i^L) \times \theta_i^L)$ is guaranteed to be provided by the job's deadline in the HC mode. ■

4.5.4.3 Schedulability tests for transition HC mode jobs

In this section, we derive the schedulability conditions for transition jobs in the HC mode. Transition HC mode jobs are released in the transition period (within $\sum_{j:1 \leq j \leq n_H} w_j$ time units from the mode switch). We further classify them into two categories depending on whether they are released before the k_i^{th} window or not. Transition jobs released before $\sum_{j:1 \leq j < k_i} w_j$ are denoted as **early transition jobs**, and the remaining transition jobs are denoted as **late transition jobs**. Note that by definition, when $k_i = n_H + 1$ all the transition jobs are in fact early transition jobs. We now derive the schedulability conditions for each of these jobs separately. Figure 4.4 shows examples of both these transition jobs.

Early transition jobs. Early transition jobs are released in the transition period and use the execution rates in the transition windows before k_i . To derive the schedulability conditions for this case, we first present a simple property of non-decreasing execution rates.

Figure 4.4: Transition jobs

Lemma 1 Suppose the execution rates in the transition windows prior to k_i satisfy the following conditions.

$$\sum_{j:1 \leq j < k_i} \theta_{i,j}^H \times w_j \geq u_i^H \times \sum_{j:1 \leq j < k_i} w_j \quad (4.6)$$

$$\forall j : 1 \leq j < k_i, \theta_{i,j}^H \leq \theta_{i,j+1}^H \quad (4.7)$$

Then, in any contiguous time period after the mode switch but immediately preceding the k_i^{th} window (say $[t, \sum_{j:1 \leq j < k_i} w_j]$ where t is a time instant relative to the mode switch instant), the total execution assigned to τ_i is at least $(\sum_{j:1 \leq j < k_i} w_j - t)u_i^H$.

Proof: Consider the time interval $[t, \sum_{j:1 \leq j < k_i} w_j]$, and let $\sum_{j:1 \leq j < p} w_j \leq t < \sum_{j:1 \leq j \leq p} w_j$ for some transition window $p < k_i$. p denotes the transition window to which time instant t belongs (see Figure 4.4(a)). We consider two cases depending on whether $\theta_{i,p}^H \geq u_i^H$ or not.

Case 1 ($\theta_{i,p}^H \geq u_i^H$): In this case, using Equation (4.7) we get that each execution rate allocated in the time interval $[t, \sum_{j:1 \leq j < k_i} w_j]$ is at least u_i^H . Therefore, it is true that the total execution assigned to τ_i is at least $(\sum_{j:1 \leq j < k_i} w_j - t)u_i^H$.

Case 2 ($\theta_{i,p}^H < u_i^H$): Consider Equation (4.6).

$$\sum_{j:1 \leq j < k_i} \theta_{i,j}^H \times w_j \geq u_i^H \times \sum_{j:1 \leq j < k_i} w_j$$

$$\begin{aligned}
& \text{(Using } \forall j : j < p, \theta_{i,p}^H \geq \theta_{i,j}^H \text{ from Equation (4.7))} \\
\Rightarrow & \sum_{j:1 \leq j \leq p} \theta_{i,p}^H \times w_j + \sum_{j:p < j < k_i} \theta_{i,j}^H \times w_j \geq \left(u_i^H \times \sum_{j:1 \leq j < k_i} w_j \right) \\
& \text{(Adding and subtracting } t \times \theta_{i,p}^H \text{)} \\
\Rightarrow & t \times \theta_{i,p}^H + \left(\sum_{j:1 \leq j \leq p} w_j - t \right) \times \theta_{i,p}^H + \sum_{j:p < j < k_i} \theta_{i,j}^H \times w_j \geq \left(u_i^H \times \sum_{j:1 \leq j < k_i} w_j \right) \\
\Rightarrow & \left(\sum_{j:1 \leq j \leq p} w_j - t \right) \times \theta_{i,p}^H + \sum_{j:p < j < k_i} \theta_{i,j}^H \times w_j \geq \left(u_i^H \times \sum_{j:1 \leq j < k_i} w_j \right) - t \times \theta_{i,p}^H \\
\Rightarrow & \left(\sum_{j:1 \leq j \leq p} w_j - t \right) \times \theta_{i,p}^H + \sum_{j:p < j < k_i} \theta_{i,j}^H \times w_j \geq u_i^H \times \left(\sum_{j:1 \leq j < k_i} w_j - t \right) \text{ (Using } \theta_{i,p}^H < u_i^H \text{)}
\end{aligned}$$

The left hand side in the above inequality denotes the exact total execution allocated to τ_i in the time interval $[t, \sum_{j:1 \leq j < k_i} w_j]$. Therefore, the lemma holds true for this case as well. ■

Condition (4.7) is necessary because the tasks that are released immediately after mode switch (within the transition window) will not be schedulable otherwise. Although it seems to be a restriction on the model, it is in fact reasonable because one can expect a higher rate assignment as more carry-over jobs complete. The following theorem presents a sufficient schedulability condition for the early transition jobs using the above lemma.

Theorem 2 *An early transition job of task $\tau_i \in \tau_H$ can meet its deadline in the HC mode if the following four conditions are met.*

$$\sum_{j:1 \leq j < k_i} \theta_{i,j}^H \times w_j \geq u_i^H \times \sum_{j:1 \leq j < k_i} w_j \quad (4.8)$$

$$\forall j : 1 \leq j < k_i, \theta_{i,j}^H \leq \theta_{i,j+1}^H \quad (4.9)$$

$$\forall j : k_i \leq j \leq n_H, \theta_{i,j}^H \geq u_i^H \quad (4.10)$$

$$\theta_i^H \geq u_i^H \quad (4.11)$$

Proof: Consider an early transition job released at some time instant t after the mode switch as shown in Figure 4.4(a). By definition, $t < \sum_{j:1 \leq j < k_i} w_j$, $t < T_i - C_i/\theta_i^L$ and the job's deadline is no earlier than $T_i - C_i/\theta_i^L$. The latter follows from the fact that the job is released after the mode switch, and hence its deadline cannot be earlier than the earliest possible deadline of a carry-over job.

Consider the time interval $[t, \sum_{j:1 \leq j < k_i} w_j]$. Using Equations (4.8) and (4.9) and Lemma 1 we get that the total execution assigned to the job in this interval is at least $(\sum_{j:1 \leq j < k_i} w_j - t) \times u_i^H$. In each succeeding window, the job is assigned an execution rate of at least u_i^H (Equations (4.10) and (4.11)). Therefore, the total execution assigned to the job between its release time and deadline is at least C_i^H , and hence the job can meet its deadline. ■

Late transition jobs. The only remaining jobs to consider are late transition jobs that are released in the transition period but no earlier than $\sum_{j:1 \leq j < k_i} w_j$. Since these jobs only use the execution rates in windows at or after k_i , Equations (4.10) and (4.11) in Theorem 2 are sufficient to guarantee the schedulability of these jobs. If each of those rates are at least u_i^H , then the late transition jobs are guaranteed to execute for C_i^H time units by their deadlines.

Thus, combining Propositions 1–3 and Theorems 1 and 2 we obtain a sufficient schedulability test for the multi-rate fluid scheduling model. Also, note that Proposition 4 is subsumed by Theorem 2. Note that, the test has polynomial time complexity because all the equations can be evaluated in time proportional to the number of tasks in τ . We now verify the schedulability for the example task set shown in Table 4.1 under the multi-rate model.

Example 5 Consider the task set τ with multi-rate assignments as shown in Table 4.1. We show that this rate and window assignment is schedulable.

We can easily check that Propositions 1–3, Equations (4.4) and (4.5) of Theorem 1 and all the equations of Theorem 2 are satisfied.

Preposition 1: for τ_1 , $0.571428 \geq 0.4$; for τ_2 $0.6 \geq 0.3$; for τ_3 $0.186766 \geq 0.1$ and for τ_4 $0.45 \geq 0.45$. *Preposition 2:* $\sum_{\tau_i \in \tau} \theta_i^L = 1.808294 \leq 2$ *Preposition 3:* $\sum_{\tau_i \in \tau_H} \theta_{i,1}^H = 2.0 \leq 2$, $\sum_{\tau_i \in \tau_H} \theta_{i,2}^H = 2.0 \leq 2$, $\sum_{\tau_i \in \tau_H} \theta_{i,3}^H = 2.0 \leq 2$ and $\sum_{\tau_i \in \tau_H} \theta_i^H = 2.0 \leq 2$.

From the definition of k_i , we have $T_1 - C_1^L/\theta_1^L = 7 - (2.8/0.571428) = 7 - 4.9 = 2.1$, $T_2 - C_2^L/\theta_2^L = 5 - (1.5/0.6) = 5 - 2.5 = 2.5$ and $T_3 - C_3^L/\theta_3^L = 35 - (3.5/0.186766) = 35 - 18.74 = 16.26$. Thus, $w_1 = 2.1$, $w_1 + w_2 = 2.5$ and $w_1 + w_2 + w_3 = 16.26$. This implies, $k_1 = 1$, $k_2 = 2$ and $k_3 = 3$.

Theorem 1. Equation (4.4): for τ_1 , $0.571428 \leq 1.0$ and $0.571428 \leq 0.7$; for τ_2 , $0.6 \leq 1.0$ and $0.6 \leq 0.8$; for τ_3 , $0.186766 \leq 0.5$. *Equation (4.5):* for τ_1 , $0.571428 \leq 0.7$; for τ_2 , $0.6 \leq 0.8$; for τ_3 , $0.186766 \leq 0.3$.

Theorem 2. Equation (4.8): for τ_1 , $1 \times 2.1 \geq 0.7 \times 2.1$; for τ_2 , $(1 \times 2.1 + 1 \times 0.4) \geq 0.8 \times 2.5$;

for τ_3 , $(0 \times 2.1 + 0.3 \times 0.4 + 0.5 \times 13.76) \geq 0.3 \times 16.26$. Equation (4.9): for τ_2 , $1 \leq 1$; for τ_3 , $0 \leq 0.3 \leq 0.5$. Equation (4.10): For τ_1 , $1 \geq 0.7$ and $0.7 \geq 0.7$; for τ_2 , $1 \geq 0.8$ and $0.8 \geq 0.8$; Equation (4.11): for τ_1 , $\theta_1^H = 0.7 = u_1^H$; for τ_2 , $\theta_2^H = 0.8 = u_2^H$; for τ_3 , $\theta_3^H = 0.3 = u_3^H$.

Now consider Equation (4.3) of Theorem 1: for τ_1 , $1 \times 2.1 \geq (4.9 - 2.8)$; for τ_2 , $(1 \times 2.1 + 1 \times 0.4) \geq (4 - 1.5)$; for τ_3 , $(0 \times 2.1 + 0.3 \times 0.4 + 0.5 \times 13.76) \geq (10.5 - 3.5)$. Thus, Equation (4.3) is also satisfied, and hence the task set is schedulable.

4.6 Properties of the multi-rate model

In this section, we derive some important properties of the proposed multi-rate fluid scheduling model and its schedulability test. A *rate and window assignment algorithm* for the multi-rate fluid scheduling model is an algorithm that assigns values for all the execution rates in the model as well as for the n_H transition window durations. We define the correctness criteria for such an algorithm in Definition 20. In Section 4.6.1 we show that the multi-rate schedulability test is equivalent to the dual-rate schedulability test (Chapter 2, Section 2.3.2, Definition 15) if we set all the transition rates to θ_i^H . Later, in Section 4.6.2, we show that the rate and window assignment algorithm has an optimal speed-up factor of $4/3$.

Definition 20 (Multi-Rate MC-Correctness) *A rate and window assignment algorithm is called **Multi-Rate MC-Correct** iff the following holds: Whenever the algorithm returns a set of execution rates ($\forall i : \theta_i^L, \theta_i^H$ and $\forall i, j : \theta_{i,j}^H$) and transition window durations ($\forall j : w_j$), these rates and window durations satisfy Propositions 1–3 and Theorems 1 and 2.*

4.6.1 Dual-rate Generalization

An interesting property of the proposed multi-rate model and schedulability test is that it generalizes the dual-rate model and schedulability test discussed in Section 2.3.2. We can obtain the dual-rate model by setting $\forall j : \theta_{i,j}^H = \theta_i^H$ for each HC task τ_i . For this case, the following lemma shows that the multi-rate schedulability test derived in Section 4.5 is equivalent to the dual-rate schedulability test. However, the equivalence proof is only valid when the conditions of the lemma hold.

Lemma 2 (Dual-rate Generalization) *If $\forall i : 1 \leq i \leq n_H, \forall j : 1 \leq j \leq n_H, \theta_{i,j}^H = \theta_i^H$ in the multi-rate fluid model, then Propositions 1–3 and Theorems 1 and 2 are satisfied if and only if*

Equations (2.4)–(2.8) hold.

Proof: The conditions for LC mode platform feasibility and LC task schedulability are identical for the dual-rate and multi-rate models (Proposition 1 and Equation (2.4), Proposition 2 and Equation (2.7)). We now consider the HC task schedulability and HC mode platform feasibility tests.

(Dual-rate $\Rightarrow$ Multi-rate): Suppose the tests are satisfied for the dual-rate model. That is, Equations (2.5), (2.6) and (2.8) hold.

By substituting Equation (2.6) in Equation (2.5) we get,

$$\begin{aligned}
 1 &\geq \frac{u_i^H - u_i^L}{\theta_i^H} + \frac{u_i^L}{\theta_i^L} && \text{Equation (2.5)} \\
 \Leftrightarrow 1 &\geq \frac{u_i^H - u_i^L}{\theta_i^H} + \frac{u_i^L}{\theta_i^H} && (\text{Using } \theta_i^L \leq \theta_i^H) \\
 \Leftrightarrow \theta_i^H &\geq u_i^H && (4.12)
 \end{aligned}$$

First we consider Theorem 2. Equation (4.11) in this theorem is identical to Equation (4.12) above. Equations (4.8), (4.9) and (4.10) are all satisfied because $\theta_{i,j}^H = \theta_i^H \geq u_i^H$ for all i and j based on the assumption of the lemma and Equation (4.12).

Next we consider Proposition 3. Equation (4.2) in this proposition is identical to Equation (2.8). Also, Equation (4.1) reduces to $\sum_{\tau_i \in \tau_H} \theta_i^H \leq m$, which is also satisfied based on Equation (2.8). Thus the proposition is satisfied.

Finally, we consider Theorem 1. Equation (4.3) in Theorem 1 is

$$\begin{aligned}
 \sum_{j:1 \leq j < k_i} \theta_{i,j}^H \times w_j + \theta_i^H \times \left(T_i - C_i^L / \theta_i^L - \sum_{j:1 \leq j < k_i} w_j \right) &\geq C_i^H - C_i^L \\
 \Leftrightarrow \theta_i^H (T_i - C_i^L / \theta_i^L) &\geq C_i^H - C_i^L \quad (\text{Using } \theta_{i,j}^H = \theta_i^H, \forall i, j) \\
 \Leftrightarrow \theta_i^H T_i (1 - u_i^L / \theta_i^L) &\geq T_i (u_i^H - u_i^L) \\
 \Leftrightarrow 1 - u_i^L / \theta_i^L &\geq \frac{u_i^H - u_i^L}{\theta_i^H} \\
 \Leftrightarrow 1 &\geq \frac{u_i^H - u_i^L}{\theta_i^H} + \frac{u_i^L}{\theta_i^L} \quad (\text{Equation (2.5)})
 \end{aligned}$$

Equations (4.4) and (4.5) reduce to $\theta_i^L \leq \theta_i^H$ using $\theta_{i,j}^H = \theta_i^H, \forall i, j$. This is identical to Equations (2.4) and (2.5).

tion (2.6). Thus, we have shown that satisfaction of the dual-rate schedulability test also implies satisfaction of the multi-rate schedulability test.

(Multi-rate $\Rightarrow$ Dual-rate): This case is trivial because Proposition 3, and Theorems 1 and 2 subsume Equations (2.5), (2.6) and (2.8). ■

4.6.2 Speed-up Optimality

In this section, we show that the rate and window assignment algorithm for the multi-rate model has an optimal speed-up factor of $4/3$. As discussed in Section 2.3.2, dual-rate assignment strategy MC-Fluid has been shown to be speed-up optimal with a speed-up factor of $4/3$ [29]. We use the generalization lemma presented in Section 4.6.1 to show that rate and window assignment algorithms for the multi-rate model satisfying some properties are also speed-up optimal. In particular, we will show that rate and window assignment algorithms for the multi-rate model that dominate MC-Fluid in terms of schedulability are also speed-up optimal with a speed-up factor of $4/3$.

Since MC-Fluid has a speed-up factor of $4/3$, this means any implicit-deadline dual-criticality MC task system satisfying the condition $\max\{U_L^L + U_H^L, U_H^H\} \leq 3/4$ and $\max_i\{u_i^L, u_i^H\} \leq 3/4$ is schedulable under MC-Fluid. Then, if some multi-rate and window assignment algorithm $\mathbb{A}$ dominates MC-Fluid, it implies that any implicit-deadline dual-criticality MC task system satisfying these conditions is also schedulable under $\mathbb{A}$. This directly implies a speed-up factor of $4/3$ for $\mathbb{A}$, because any feasible MC task system will satisfy the above conditions when the speed of each processing core is increased by a factor of $4/3$. Based on this intuition, the following lemma derives properties for $\mathbb{A}$ that ensures $\mathbb{A}$ dominates MC-Fluid.

Lemma 3 (Dominance and Speed-up Factor) *Suppose a rate and window assignment algorithm $\mathbb{A}$ for the multi-rate fluid model is guaranteed to return some feasible assignment as long as there is at least one assignment satisfying,*

- $\forall i : 1 \leq i \leq n_H, \forall j : 1 \leq j \leq n_H, \theta_{i,j}^H = \theta_i^H,$
- $\forall j : 1 \leq j \leq n_H, w_j = 0,$ and
- *Propositions 1–3 and Theorems 1 and 2.*

Then Algorithm $\mathbb{A}$ dominates MC-Fluid and has a speed-up factor of $4/3$.

Proof: Suppose an implicit-deadline dual-criticality MC task system is feasible under MC-Fluid. Then, let θ_i^L and θ_i^H denote the execution rates for each task $\tau_i \in \tau_H$ assigned by MC-Fluid that satisfies Equations (2.4)–(2.8). Now consider the multi-rate assignment $\forall i : 1 \leq i \leq n_H, \forall j : 1 \leq j \leq n_H, \theta_{i,j}^H = \theta_i^H, \forall j : 1 \leq j \leq n_H, w_j = 0$ and θ_i^L identical to the dual-rate assignment. From Lemma 2 we know that this rate assignment satisfies Propositions 1–3 and Theorems 1 and 2. Thus, there is at least one rate and window assignment that satisfies all the above three conditions of the lemma. From the assumption of the lemma, we then know that algorithm $\mathbb{A}$ is guaranteed to return a feasible multi-rate and window assignment. This shows that algorithm $\mathbb{A}$ dominates MC-Fluid in terms of schedulability. Since MC-Fluid has a known speed-up factor of $4/3$, by definition algorithm $\mathbb{A}$ also has a speed-up factor of $4/3$. ■

It has been shown that no non-clairvoyant scheduling algorithm can have a speed-up factor lower than $4/3$ for scheduling dual-criticality implicit-deadline sporadic task systems on multi-core platforms [62]. The speed-up optimality of the multi-rate and window assignment algorithm $\mathbb{A}$ with a speed-up factor of $4/3$ follows by combining Lemma 3 and the fact that no non-clairvoyant algorithm can have a speed-up factor smaller than $4/3$.

4.7 Mapping to non-fluid platform

The multi-rate fluid scheduling model, similar to other fluid models, assumes that a processing core can be fractionally assigned to tasks. Since this is not possible on a real (non-fluid) platform, it is important to map the fluid execution rates to a non-fluid scheduling policy, ideally without any loss in schedulability. In [28], the dual-rate fluid scheduling model has been successfully mapped without any loss in schedulability to the DP-Fair (short for Deadline-Partitions Fair) non-fluid scheduling algorithm [14]. In this section, we now show that the multi-rate fluid scheduling model can also be similarly mapped to DP-Fair without any loss in schedulability.

In the classic real-time multi-core scheduling, DP-Fair scheduling algorithm has been used to map a fluid execution model with single execution rate for each task to a non-fluid schedule without any loss in schedulability [14]. The main intuition behind this mapping is as follows. The entire scheduling window is partitioned based on job deadlines. Between any two consecutive job deadlines, it is ensured that the total allocation to a task is equal to the length of the partition multiplied by its assigned execution rate. Thus, tasks are guaranteed processing core allocations proportional to their execution rates at every job deadline, and hence schedulability is preserved. Between two consecutive job deadlines, the processing core share allocated to each task can be

scheduled using any non-fluid optimal scheduling policy such as McNaughton's algorithm [91]. To handle sporadic job releases, the allocations in each partition are scheduled using a work-conserving algorithm, and they are re-computed upon job releases. Note that this scheduling policy can be implemented fully online by computing allocations for pending jobs between the current time instant (coinciding with a job release or deadline) and the next earliest job deadline.

In the multi-rate model, apart from job deadlines, another important event is the mode switch instant. A mode switch is triggered when a HC job executes for its LC execution budget (C_i^L) and does not signal completion. For correctness of the multi-rate schedulability test derived in Section 4.5, it is essential to ensure that each HC job of a task $\tau_i \in \tau_H$ completes its LC execution C_i^L no later than C_i^L / θ_i^L time units from its release. This is necessary to guarantee the property that any job deadline of task τ_i in HC mode is no earlier than $T_i - C_i / \theta_i^L$ time units from the mode switch, which is used in Theorems 1 and 2. Hence, when mapping the multi-rate model to DP-Fair scheduling, in addition to ensuring task allocation fairness at job deadlines, we also need to ensure this fairness at *worst-case mode switch instants* for each HC job. That is, we need to ensure fairness at C_i^L / θ_i^L time units from each job release instant for each HC task τ_i .

After the mode switch, one simple way to map the multi-rate model to DP-Fair scheduling is to ensure allocation fairness at the boundary of each transition window j ($1 \leq j \leq n_H$), in addition to job deadlines. Of course, this is not necessary, and schedulability would be guaranteed even if fairness is only ensured at job deadlines. Suppose a partition (interval between two consecutive job deadlines or between the mode switch and the earliest job deadline) spans more than one transition window. Let $[t_1, t_2]$ denote this partition, $j_1 (\leq n_H)$ denote a transition window that contains t_1 , and $j_2 (\leq n_H)$ denote another transition window that contains t_2 . Then, for each HC task τ_i it is sufficient to ensure that its total allocation in this partition is equal to $(\sum_{1 \leq j \leq j_1} w_j - t_1) \theta_{i,j_1}^H + (\sum_{j_1 < j < j_2} w_j) \theta_{i,j}^H + (t_2 - \sum_{1 \leq j < j_2} w_j) \theta_{i,j_2}^H$.

Thus, the multi-rate MC fluid scheduling model can be mapped to the DP-Fair scheduling algorithm using the following strategy.

Definition 21 (DP-Fair Mapping) *Given a rate and window assignment in the multi-rate model, consider the following task allocations.*

- **LC mode mapping:** *Partition the scheduling window based on job deadlines and worst case mode switch instants (C_i^L / θ_i^L time units from job release for each HC task τ_i). In each partition p having length L_p , allocate an execution of $\theta_i^L \times L_p$ time units for each task τ_i that has an unfinished job at the beginning of the partition.*

- **Transition period mapping:** After mode switch drop all the jobs of LC tasks. Partition the transition period (within $\sum_{1 \leq j \leq n_H} w_j$ time units from the mode switch) based on HC job deadlines. Consider a partition p spanning the interval $[t_1, t_2]$, such that $j_1(\leq n_H)$ denotes a transition window that contains t_1 , and $j_2(\leq n_H)$ denotes another transition window that contains t_2 . Allocate an execution of $(\sum_{1 \leq j \leq j_1} w_j - t_1)\theta_{i,j_1}^H + (\sum_{j_1 < j < j_2} w_j)\theta_{i,j}^H + (t_2 - \sum_{1 \leq j < j_2} w_j)\theta_{i,j_2}^H$ time units for each HC task τ_i that has an unfinished job at t_1 .
- **HC mode mapping:** After $\sum_{1 \leq j \leq n_H} w_j$ time units from the mode switch, partition the scheduling window based on HC job deadlines. In each partition p having length L_p , allocate an execution of $\theta_i^H \times L_p$ time units for each HC task τ_i that has an unfinished job at the beginning of the partition.

In each partition, schedule the task allocations using any work-conserving optimal algorithm.

The following lemma records the fact that the above mapping is optimal, in the sense that there is no loss in schedulability.

Lemma 4 Consider a set of execution rates and window durations that satisfy Propositions 1–3 and Theorems 1 and 2. If these rates are mapped to the DP-Fair scheduling algorithm using Definition 21, then in the resulting schedule all job deadlines are met.

Proof: In the LC mode, each job of a task τ_i is guaranteed to receive execution proportional to its assigned rate θ_i^L within its deadline. This follows from Proposition 2 and the fact that allocations in each partition are scheduled using a work-conserving optimal algorithm. Then, using Proposition 1 we can conclude that all the job deadlines in the LC mode are met.

Each job of a HC task τ_i is guaranteed to receive at least C_i^L time units of execution within C_i^L / θ_i^L time units from its release. Therefore, the deadline of any job of τ_i in the HC mode is no earlier than $T_i - C_i^L / \theta_i^L$ time units from the mode switch instant.

In the HC mode, each job is guaranteed to receive execution proportional to the assigned rates within its deadline (using Proposition 3 and the fact that allocations are scheduled using a work-conserving optimal algorithm). Therefore, using Theorems 1 and 2 we can conclude that all job deadlines in the HC mode will be met. ■

4.8 Multi-rate Assignment Strategy

In this section, we present an algorithm called *SOMA* (short for Speed-up Optimal Multi-rate Assignment) to determine the execution rates and window durations for the multi-rate model. A convex optimization based solution, similar to the dual-rate algorithm MC-Fluid, is desirable because it can optimize the assignments. However, one major hurdle towards using an optimization framework for the multi-rate model is that the earliest completion window parameters, k_i s in Definition 19, control the number of terms involved in the summations used in the schedulability test. Since k_i s are dependent on the execution rates and window durations, this would imply that the constraints of the optimization problem are no longer fixed. To address this issue, we first use a simple heuristic to fix these k_i s and then formulate the optimization problem.

Algorithm 1 presents SOMA. Since it is sufficient for all jobs of LC tasks to execute at their minimum required rate, we assign $\theta_i^L = u_i^L$ for all $\tau_i \in \tau_L$. The algorithm first sorts all the HC tasks in increasing order of the parameter $T_i - C_i^L/u_i^H$. Note that u_i^H denotes the maximum possible rate that a HC task τ_i can be assigned in the LC mode (easily seen by combining equations in Theorems 1 and 2). Then $T_i - C_i^L/u_i^H$ denotes the earliest completion time for any carry over job of this task, assuming the maximum possible rate in the LC mode. Hence, in any feasible assignment, an HC task τ_i with a smaller $T_i - C_i^L/u_i^H$, will most likely end up with a smaller k_i when compared to a HC task τ_j with a larger $T_j - C_j^L/u_j^H$. We use this intuition to fix the values for k_i s, and thus address the challenge discussed above. In spite of this restriction, we show that SOMA has a speed-up factor of 4/3 (Lemma 5 below) and has comparable performance to a brute-force technique that tries all the $n_H!$ possible combinations for the k_i s (Section 4.9).

Algorithm 1 SOMA rate and window assignment strategy

Input: τ, m

- 1: For each $\tau_i \in \tau_L$ assign $\theta_i^L = u_i^L$.
 - 2: Sort τ_H in increasing order of the parameter $T_i - (C_i^L/u_i^H)$.
 - 3: Solve the optimization problem in Definition 22, assuming task indices are sorted based on the above sorting order.
 - 4: **if** Optimization returns an assignment **then**
 - 5: **if** $\sum_{\tau_i \in \tau} \theta_i^L \leq m$ **then**
 - 6: Declare Success
 - 7: **else**
 - 8: Declare Failure
 - 9: **end if**
 - 10: **else**
 - 11: Declare Failure
 - 12: **end if**
-

SOMA then solves the convex optimization problem given in Definition 22. Using a fixed value for all the k_i s ($k_i = i$), where we assume task indices are sorted based on increasing value of the parameter $T_i - (C_i^L/u_i^H)$, this optimization encodes Propositions 1 and 3 and Theorems 1 and 2 of the schedulability test in its constraints. Its objective is to minimize the total LC mode execution rates. If it returns a feasible assignment for the rates and window durations, we check whether this returned assignment satisfies the remaining condition for schedulability, *i.e.*, Proposition 2.

Definition 22 (Multi-rate convex optimization) Suppose $k_i = i$ for each HC task $\tau_i \in \tau_H$.

$$\begin{aligned}
& \text{minimize} && \sum_{\tau_i \in \tau_H} \theta_i^L \\
& \text{subject to,} && \\
& && \text{Propositions 1 – 3} \\
& && \text{Theorems 1 and 2} \\
& && \forall i : 1 \leq i \leq n_H, \forall j : 1 \leq j \leq n_H, \theta_{i,j}^H \leq 1 \\
& && \forall i : 1 \leq i \leq n_H, \theta_i^H \leq 1 \\
& && \forall j : 1 \leq j \leq n_H, w_j \geq 0
\end{aligned}$$

It is easy to see that *SOMA* is a *Multi-Rate MC-Correct* assignment strategy. Any rate and window assignment returned by SOMA satisfies Propositions 1-3 and Theorems 1 and 2 because of the constraints of the optimization problem as well as the check in Step 5 of Algorithm 1. The following lemma shows that SOMA has a speed-up factor of 4/3.

Lemma 5 *Algorithm SOMA has a speed-up factor of 4/3.*

Proof: From Lemma 3 we know that any algorithm that always return some feasible assignment if there exists an assignment satisfying the following three conditions has a speed-up factor of 4/3.

1. $\forall i : 1 \leq i \leq n_H, \forall j : 1 \leq j \leq n_H, \theta_{i,j}^H = \theta_i^H$,
2. $\forall j : 1 \leq j \leq n_H, w_j = 0$, and
3. Propositions 1–3 and Theorems 1 and 2.

To prove this lemma, it is then sufficient to show that such an assignment is considered by SOMA. Observe that the boundary constraints on execution rates and window durations in the optimization problem allow an assignment that satisfies the above three conditions.

The constraints of the optimization problem exactly encode Propositions 1–3 and Theorems 1 and 2 of the schedulability test. Now we show that for an assignment satisfying the above conditions, the earliest completion window parameter k_i is irrelevant.

In Theorem 1, Equation (4.4) is subsumed by Equation (4.5). That is, when condition 1 holds $\forall j : k_i \leq j \leq n_H, \theta_{i,j}^L \leq \theta_{i,j}^H$ is subsumed by $\theta_i^L \leq \theta_i^H$.

Also, Equation (4.3) reduces to $\theta_i^H (T_i - C_i/\theta_i^L) \geq C_i^H - C_i^L$. Substituting $R_i = \theta_i^H$ (by condition 1) and condition 2 in Equation (4.3) we get,

$$\begin{aligned} \sum_{j:1 \leq j < k_i} \theta_{i,j}^H \times w_j + R_i \times \left(T_i - C_i^L/\theta_i^L - \sum_{j:1 \leq j < k_i} w_j \right) &\geq C_i^H - C_i^L \\ \Rightarrow \theta_i^H (T_i - C_i/\theta_i^L) &\geq C_i^H - C_i^L \end{aligned}$$

Therefore, Theorem 1 is independent of k_i values.

Similarly, in Theorem 2, Equation (4.10) is subsumed by Equation (4.11). That is, $\forall j : k_i \leq j \leq n_H, \theta_{i,j}^H \geq u_i^H$ becomes $\theta_i^H \geq u_i^H$ when condition 1 holds.

Equation (4.9), $\forall j : 1 \leq j < k_i, \theta_{i,j}^H \leq \theta_{i,j+1}^H$, is satisfied because all the transition window rates for each task are identical (*i.e.*, condition 1), and Equation (4.8) is also satisfied because both the sides equate to 0 (since $w_j = 0$, by condition 2). Thus, even this theorem is independent of k_i values.

Hence the k_i values are irrelevant for an assignment that satisfies the above conditions. Thus, any k_i values, including the ones fixed by SOMA, can be used in the optimization problem.

Thus, the optimization problem is guaranteed to find the assignment that satisfies the above three conditions. This combined with the check for Proposition 2 in Step 5 of SOMA proves the lemma. ■

Algorithm SOMA has a speed-up factor of $4/3$ by Lemma 5, and by the argument that no non-clairvoyant algorithm can have a speed-up factor smaller than $4/3$ proves the speed-up optimality of SOMA.

Algorithm Complexity. SOMA can determine the values for k_i s in linearithmic time ($O(n_H \log(n_H))$). The optimization has $n_H^2 + 3n_H$ real variables, n_H integer variables and $9 \times n_H$ constraints. Hence, the number of variables and constraints is polynomially bounded in the number of HC tasks in the system. The complexity of SOMA is therefore bounded by the complexity of solving the convex optimization problem. By replacing C_i^L / θ_i^L in the optimization problem with a variable x_i , it is easy to see that the problem reduces to a convex optimization problem with objective of the form $\sum_i 1/x_i$ and all linear constraints.

4.9 Experimental Evaluation

In this section, using the randomly generated synthetic task sets we evaluate the schedulability performance of SOMA. We compare it with existing multi-core MC scheduling algorithms that have known speed-up factors. These algorithms include GLO-EDF_VD [26], PAR-EDF_VD [33] and MC-Fluid [28].

- GLO-EDF_VD is a global scheduling algorithm based on EDF-VD and its speed-up factor is no larger than $\sqrt{5} + 1$.
- PAR-EDF_VD is a partitioned scheduling based on EDF-VD and its speed-up factor is no larger than $8/3$.
- MC-Fluid is the optimal dual-rate fluid scheduling algorithm with an optimal speed-up factor of $4/3$. Since MC-Fluid is known to be dual-rate optimal we do not compare with the sub-optimal dual-rate strategy MCF [29].

We compare with only these algorithms because our focus is on designing an algorithm that has a worst-case performance guarantee.

4.9.1 Task Set Generation

We use the task set generator proposed in [92] to generate the task sets for our experiments. The task parameters used in our generator are described as follows:

- $m \in \{2, 4, 8\}$ denotes the total number of cores.
- $u_{min} = 0.001$ denotes the minimum individual task utilization.

- $u_{max} = 0.99$ denotes the maximum individual task utilization.
- $U_H^H/m \in [0.1, 0.15, \dots, 1.0]$ denotes the normalized system utilization in the HC mode.
- $U_H^L/m \in [0.05, 0.10, \dots, U_H^H/m]$ denotes the normalized system utilization of HC tasks in the LC mode.
- $U_L^L/m \in [0.05, 0.10, \dots, 1 - U_H^L/m]$ denotes the normalized system utilization of LC tasks in the LC mode.
- $P_H \in [0.1, 0.2, \dots, 0.9]$ denotes the percentage of HC tasks in the system.
- T_i , the period of a task τ_i is drawn uniformly at random from $[10, 100]$.
- The utilization values u_i^L and u_i^H are computed using standard techniques *BoundedUniform* [92] and *MRandFixedSum* [93].
- Execution budgets C_i^L and C_i^H are derived as $u_i^L * T_i$ and $u_i^H * T_i$ respectively.

Total number of tasks is lower bounded by $m + 1$ and upper bounded by $10 * m$. Total number of HC tasks in the system is lower bounded by $m + 1$ and upper bounded by $3 * m$. Note that, the performance of SOMA is identical to MC-Fluid for values of $n_H \leq m$. This is expected because if such a task set is dual-rate infeasible, it means that even allocating the maximum rate of 1 to each task in the HC mode is not sufficient. Using the above parameters we generate at least 1000 task sets for each value of total normalized utilization U_B ($U_B = \max(\frac{U_H^L + U_L^L}{m}, \frac{U_H^H}{m})$) and m . A larger U_B value denotes the task set is harder to schedule. We consider $U_B \in [0.50, 0.55, \dots, 1.0]$. For each such successfully generated task set we evaluate the performance of the SOMA with other multi-core MC scheduling algorithms with known the speed-up factor.

4.9.2 Evaluation of Overall Schedulability Performance

We compare the overall schedulability performance of the algorithms with known speed-up factors in Figure 4.5. We plot the acceptance ratio, *i.e.*, the fraction of generated task sets that are deemed schedulable, of the algorithms versus total normalized utilization U_B for varying $m \in [2, 4, 8]$. As can be seen, SOMA clearly dominates all the other existing multi-core MC scheduling algorithms. In particular, it significantly outperforms GLO-EDF_VD and PAR-EDF_VD, and also has better schedulability performance than MC-Fluid. In addition, the perfor-

mance gap between SOMA and MC-Fluid increases with m at high U_B values mainly because the number of tasks in the task sets are also higher.

(a) $m = 2$

(b) $m = 4$

Figure 4.5: Evaluation of algorithms with known speed-up factor for varying m

Since, SOMA significantly outperforms global and partitioned variants, in the remainder of this section, we take a closer look at the performance of SOMA in comparison to MC-Fluid.

4.9.3 Evaluation of MC Fluid Scheduling Algorithms

In this section, we show the benefit of the multi-rate fluid model over the dual-rate model. Since an exact feasibility test for MC task systems is not known ($U_B \leq 1$ is only necessary but not sufficient), it is difficult to exactly characterize the performance gap between SOMA and MC-Fluid. Nevertheless, to provide some insights into the significance of this gap, we show the percentage of *dual-rate infeasible task sets* that are deemed to be schedulable under SOMA in Figure 4.6.

Since MC-Fluid is known to be speed-up optimal with the speed-up factor of $4/3$, there are very few feasible task sets with $U_B \leq 0.75$ that are unschedulable under MC-Fluid. As SOMA also has an optimal speed-up factor, there is no difference in performance between SOMA and MC-Fluid for $U_B \leq 0.75$. Therefore, both algorithms have optimal performance for $U_B \leq 0.75$. As expected, we can see that SOMA performs better for $U_B \geq 0.80$. For $m = 2$, as much as

Figure 4.6: Percentage of task sets that are schedulable under multi-rate but not under dual-rate

35.8% more task sets are deemed to be schedulable under SOMA. Thus, there are significant number of task sets that are schedulable under SOMA but not under MC-Fluid (about 358 out of 1000 when $U_B = 0.80$). We believe this is a significant improvement over the state-of-art algorithm with the known speed-up factor. The performance of SOMA initially increases and then decreases as U_B increases. The reason being these task sets are harder to schedule compared to those with lower U_B values. Nevertheless, SOMA performs better than MC-Fluid at high U_B values.

The reason why this performance gap is not explicitly visible in Figure 4.5 is that because we plot the percentage of schedulable task sets inclusive of both the dual-rate feasible and dual-rate infeasible task sets. Whereas, in Figure 4.6 we only consider the dual-rate infeasible task sets. For $m = 2$, when $U_B = 0.80$ the acceptance ratio of SOMA and MC-Fluid in Figure 4.5 is 97.1% and 95.5% respectively. The performance gap between SOMA and MC-Fluid seems to be only 1.6% ($97.1\% - 95.5\%$). But, 1.6% out of the 4.5% ($100\% - 95.5\%$) dual-rate infeasible task sets are schedulable by SOMA. This translates to a performance improvement of 35.5% ($1.6/4.5 \times 100\%$) over the MC-Fluid which is consistent with Figure 4.6.

It is also important to note that since exact feasibility tests are not known for mixed-criticality systems, the task sets generated in the evaluation are not necessarily all feasible. In fact, it is

reasonable to expect that the number of feasible task sets among those generated in the evaluation would decrease with increasing m and utilization. For $m = 8$, the performance gap between SOMA and MC-Fluid increases gradually as U_B increases. This is expected because the number of HC tasks in the task sets are also higher for the same value of U_B .

4.9.4 Performance Evaluation for Varying Task Parameters

To evaluate the performance of SOMA for varying values of task set parameters P_H and u_{max} , we plot the weighted acceptance ratio (*i.e.*, acceptance ratio weighted with U_B) of the algorithm for varying $m = \{2, 4, 8\}$ in Figures 4.7- 4.9. Weighted acceptance ratio is defined as

$$WAR(\mathbb{S}) = \frac{\sum_{U_B \in \mathbb{S}} (AR(U_B) \cdot U_B)}{\sum_{U_B \in \mathbb{S}} U_B} \quad (4.13)$$

where $\mathbb{S}$ is the set of U_B values and $AR(U_B)$ is the acceptance ratio for a specific value of the system utilization U_B . This metric gives more importance to task sets that are in general harder to schedule *i.e.*, task sets with higher U_B values. As a result, this is an effective metric to combine the impact of different task parameters on the acceptance ratio.

(a) $m = 2$

Figure 4.7: Evaluation of fluid scheduling algorithms for varying n_H

Varying number of HC tasks. An important factor in MC fluid scheduling is the number of HC tasks (n_H). Therefore, in Figure 4.7, we compare the weighted acceptance ratios of SOMA and MC-Fluid for varying n_H and m . As n_H increases the performance gap between SOMA and MC-Fluid also increases. This is expected because for larger n_H at the same U_B value, there is more flexibility in assigning multiple execution rates in the HC mode. Although the performance gap increases, the performance of SOMA increases initially and decreases as n_H increases. This is expected because as the number of HC tasks in a task set increases, these tasks become hard to schedule in the HC mode.

Varying percentage of HC tasks. We compare the schedulability performance of SOMA and MC-Fluid for varying P_H values in Figure 4.8(a). We plot the weighted acceptance ratios of the algorithms. As shown, the performance gap increases as the number of HC tasks in a task set increases, indicating that SOMA performs better when there are more number of HC tasks in the system. The reason being the total LC mode rate for the HC tasks is reduced by assigning multiple rates to these tasks in the HC mode. This reduction is useful only when there are LC tasks that can benefit from it. At high P_H values, there are very few LC tasks and as a result, all these tasks will be benefitted.

Varying individual task utilization. We compare the performance of SOMA and MC-Fluid for varying $\max\{u_i^H\}$ values in Figure 4.8(b), where we plot the weighted acceptance ratios of the algorithms. As shown, the performance of both algorithms decreases with $\max\{u_i^H\}$. This is reasonable because typically the number of tasks in each task set reduces with increasing $\max\{u_i^H\}$ and as shown in Section 4.9, SOMA tends to perform better with increasing n_H .

When $\max\{u_i^H\}$ is large, there is a significant performance gap between SOMA and MC-Fluid. This is because in such task systems there are typically more number of heavy utilization tasks. As a result, the minimum required HC mode execution rate under MC-Fluid is high and hence it does not have much flexibility. However, under SOMA, some execution rates in the HC mode can be much lower than u_i^H as long as the average rate over all the transition windows is reasonably high. This flexibility in rate assignment is a key property of SOMA and its benefit is clearly visible for task sets with high $\max\{u_i^H\}$ values.

Varying normalized system utilization. We compare the performance of SOMA and MC-Fluid for varying U_L^L/m , normalized system utilization values of LC tasks. We plot the weighted

(a) Varying percentage of HC tasks (P_H) (b) Varying maximum HC utilization of HC tasks ($\max\{u_i^H\}$)

Figure 4.8: Evaluation of fluid scheduling algorithms for varying task parameters

Figure 4.9: Evaluation of fluid scheduling algorithms for varying task set parameter (U_L^L/m)

acceptance ratios of the algorithms for varying U_L^L/m values with $U_B = 0.95$ in Figure 4.9. The performance gap widens as U_L^L/m increases, indicating that SOMA performs better when the system utilization of LC tasks is high. This is reasonable because SOMA tries to minimize the total LC mode execution rate of HC tasks and as a result, this reduction becomes more useful as U_L^L/m increases.

Thus, the results from our experiments show that SOMA outperforms all the existing algorithms with known speed-up factors. The overall performance gap between SOMA and MC-Fluid increases with utilization. Further, we have also shown that this gap is significant in terms of the percentage of dual-rate infeasible task sets that SOMA can schedule, and that the gap widens with increasing number of HC tasks in the system.

4.10 Efficacy of the multi-rate fluid scheduling

In Section 4.1, we discussed that scheduling algorithms with good theoretical performance such as low speed-up factor and good performance in experimental evaluation are usually preferred. Speed-up factor is a good metric to compare the worst-case performance of different scheduling algorithms. For scheduling algorithms such as partitioned scheduling algorithms that

are based on heuristics, it is hard to determine these speed-up factors. The performance of these algorithms is often evaluated based on experimental evaluation. Hence, in the initial part of this section, we present a comprehensive evaluation of the multi-rate fluid model against the dual-rate model in terms of both theoretical and experimental performance. Then, we also show that the multi-rate model is a sub-optimal MC scheduling algorithm *i.e.*, there exists a feasible task system that is not schedulable under multi-rate fluid scheduling. In the latter part of this section, we discuss some of the challenges and considerations for implementing the proposed multi-rate fluid model in a real-time operating system (RTOS).

4.10.1 Comparative evaluation: dual-rate vs multi-rate fluid model

The dual-rate fluid model is shown to be speed-up optimal [29] and has good performance in experimental evaluation for multi-core implicit-deadline dual-criticality MC task systems [28]. Though the dual-rate model is optimal in terms of speed-up factor, it is not optimal in terms of schedulability [78]. There are several feasible task systems that are deemed unschedulable by MC-Fluid [28] - the optimal dual-rate assignment algorithm. Thus, we propose SOMA the multi-rate and window assignment algorithm which has an optimal speed-up factor and performs better than MC-Fluid in experimental evaluation (refer Section 4.9). In fact, our experimental evaluation together with Lemma 3 also shows that SOMA strictly dominates MC-Fluid in terms of schedulability. That is, all task systems deemed schedulable by MC-Fluid are also deemed so by SOMA, and there exist task systems that are not schedulable by MC-Fluid but are deemed so by SOMA.

To provide some insights on the complexity of these two rate assignment algorithms - SOMA and MC-Fluid - we compare their offline and runtime complexity. The rate assignment in both these algorithms is done offline when determining the execution rates. The offline complexity of MC-Fluid rate assignment algorithm is $O(n_H^2)$. Whereas, the offline complexity of SOMA is larger compared to MC-Fluid because the complexity of SOMA is bounded by the complexity of solving the optimization problem defined in Definition 22. For details on the complexity of SOMA refer to the complexity discussion above. During runtime, the scheduler executes the task with the assigned execution rates. The runtime scheduling mechanism of SOMA is quite similar to MC-Fluid because the n_H window durations that are determined offline remain fixed during runtime. The runtime complexity of SOMA is not significantly high compared to dual-rate model because the only additional overhead incurred is in the transition period (n_H

windows) when the mode switch is triggered. Essentially, upon mode switch in the dual-rate model, HC tasks switch their execution rate at most once. Whereas, in the case of multi-rate model, each HC task switches its execution rate at most $n_H + 1$ times.

Although the multi-rate model is speed-up optimal and has the best schedulability among existing multi-core MC scheduling algorithms, the multi-rate model is still not an optimal MC scheduling algorithm in terms of schedulability. In the next section, we show that the multi-rate model is a sub-optimal MC scheduling algorithm.

4.10.2 Sub-optimality of the multi-rate fluid model

The design of optimal (in terms of schedulability) algorithms for MC systems is very unlikely because the MC scheduling problem is known to be NP-Hard. The hardness arises from the fact that the mode switch instant is not known a priori. In the case of MC fluid scheduling, the execution rates of the tasks determine the mode switch instant, *i.e.*, the time instant when an HC task overruns its LC execution budget. Hence, we try to determine whether multi-rate fluid scheduling can optimally schedule the dual-criticality MC system given a mode switch instant. Under the MC fluid scheduling, the mode switch instant corresponds to the virtual deadline in the LC mode *i.e.*, $D_i^L = C_i^L / \theta_i^L$. If any HC task does not complete its LC execution budget within its virtual deadline, then the mode switch is triggered.

With the help of an example, assuming the virtual deadlines of HC tasks are known, we show that the dual-criticality MC task system is not schedulable under the multi-rate model but there exists a feasible MC schedule for that system. We consider a simple dual-criticality MC system with two tasks scheduled on a single-core platform. It is important to note that although the multi-rate model was derived for multi-core platforms, it can be applied to single-core platforms as well. Since fluid scheduling allows fraction processing core allocation, as long as the total execution rates are no larger than the total processing capacity of the platform fluid scheduling is still valid.

Example 6 *Consider an example dual-criticality MC task system scheduled on a single-core platform as shown in Table 4.2. Let us suppose we consider an implicit-deadline period task system. Then, under the fluid scheduling model, the worst-case mode switch instant occurs when virtual deadlines of all HC tasks align together.*

We show that the given task system is unschedulable under the multi-rate model, and there

exists a feasible schedule using a different fluid scheduling.

Table 4.2: A feasible task system deemed unschedulable by the multi-rate model

	T_i	C_i^L	C_i^H	χ_i	u_i^L	u_i^H	D_i^L
τ_1	10	1	4	HC	0.1	0.4	5
τ_2	10	1.5	5.2	HC	0.15	0.52	2

(a) Multi-rate fluid model

(b) Extended fluid model

Figure 4.10: Schedule of multi-rate infeasible task system

The schedule and the maximum execution rates assigned for the example task system under the multi-rate model is shown in Figure 4.10(a). Although the execution rates assigned satisfy the platform feasibility conditions, the execution rates are not sufficient to schedule the early release jobs. Suppose that τ_2 releases a job immediately after the mode switch by τ_1 . Then, for this job of τ_2 to be schedulable the sum product of window duration and execution rates assigned in each of these windows must be greater than its HC execution budget. As it can be seen, the rates assigned are not sufficient ($\sum w_j \cdot \theta_{2,j}^H < C_2^H$ i.e., $(5 \cdot 0.4) + (3 \cdot 0.6) + (2 \cdot 0.6) = 5 < 5.2$) to schedule jobs of this type under the multi-rate model. Thus, the given dual-criticality MC task system is not schedulable under the multi-rate model with the assumed virtual deadline.

In Figure 4.10(b), we show another possible execution rate assignment different from the

multi-rate assignment, called the extended fluid model, where each HC task is executed with more than n_H execution rates in the HC mode. We can see that the early release job of τ_2 is now schedulable under the extended fluid model ($\sum w_j \cdot \theta_{2,j}^H \geq C_2^H$ i.e., $(5 * 0.4) + (3 * 0.7) + (2 * 0.5571) = 5.21 > 5.2$). As shown, the assigned execution rates are feasible and the task system is deemed schedulable.

Thus, although the multi-rate fluid scheduling model has better schedulability performance it is still not optimal in terms of schedulability.

4.10.3 Challenges and considerations for implementation

The analysis and evaluation of the multi-rate fluid scheduling presented in the previous sections do not consider system overheads (such as task release, scheduling and context switching overhead) into account. As a result, the actual efficiency of the algorithm and applicability of the results is still unknown. In order to draw some real conclusion, multi-rate fluid scheduling has to be implemented in an RTOS environment. In this section, we explore the question of how best to implement the multi-rate fluid model. For multi-rate fluid scheduler to be viable, we list some key overhead related requisites that must be met.

The foremost challenge is that fluid scheduling cannot be realized on a real hardware platform since real processing core can execute only one task at a time. To address this problem, in Section 4.7, we presented a technique to convert multi-rate fluid schedules to equivalent non-fluid schedules. Although the technique presented allows discrete-time executions it allows mode switch at real-number time points. This gives rise to very high overhead if implemented in RTOS. Thus, the first step would be to determine a discrete multi-rate algorithm that allows both discrete-time mode switches and discrete-time executions. In the recent past, a similar discrete-time algorithm has been developed for the dual-rate fluid model [66]. However, there is a certain loss in schedulability performance for such a discrete-time algorithm. Consequently, there will also be a loss in schedulability for the equivalent discrete-time multi-rate algorithm.

The rate and window assignment for the multi-rate fluid model is done offline. Each rate and window assignment in fluid schedule translates to a fixed execution time slot in the non-fluid schedule. During run-time, the execution time slots remain unchanged. The role of the scheduler is to execute the jobs for the specified time slots and decide which job to execute based on the system operating condition. Since the multi-rate model has an additional n_H windows

after mode switch, the run-time scheduler has to be invoked n_H times to update the execution of active jobs. This is an additional but inevitable overhead compared to the dual-rate algorithm.

The scheduler for multi-rate scheduling algorithm has three components: *run-time dispatcher*, *execution manager* and *ready-queue manager*. The role of the dispatcher is to determine which active job to execute at each time instant. The dispatcher needs to take scheduling decisions when a new job is released or when there is an idle core upon completion of some jobs or when the assigned execution time slot of a job expires. The execution manager keeps track of events related to execution time slot assignment (or core allocation) such as mode switch triggering, carry-over jobs completion, etc. The execution manager is responsible for triggering an event if the job's execution (rate assignment) has to be changed. A budget enforcement timer is required to keep track of the actual job executions. It also maintains a data structure to keep track of the execution time slots of each HC job. Upon completion of a carry-over job on any of the processing core, the execution manager has to be invoked to update the job executions. The ready-queue manager maintains the ready queue by inserting or deleting active or completed job to and from the ready queue. It is also responsible for managing the execution of jobs (ensuring the rate assignment is met) by responding to the event triggered by the execution manager. There is an inherent overhead associated with each of the three components.

Thus, the multi-rate fluid scheduling sustains from challenges in translating fluid to discrete-time non-fluid schedules. Moreover, there is likely a schedulability loss as in the dual-rate fluid scheduling. Unlike dual-rate scheduling, there is an additional scheduling overhead incurred during the transition period of the carry-over jobs. Therefore, the amalgamated effect of these overheads has to be studied to evaluate the efficacy of multi-rate fluid scheduling.

4.11 Conclusions

Global MC scheduling algorithms suffer from poor schedulability. To address this issue, dual-rate fluid scheduling model was proposed. Although the dual-rate model improves the schedulability performance, it is still not capable of scheduling a large class of task systems. In particular, it is very inefficient at high utilization ranges ($> 3/4$). To overcome the schedulability issue, in this chapter, we presented the multi-rate fluid scheduling model for implicit-deadline dual-criticality MC systems.

Under the multi-rate model, each HC task is executed with multiple execution rates in the

transition period after the mode switch. We derived the sufficient schedulability test for the proposed model. We showed that the multi-rate model is a generalization of the dual-rate model, and has the optimal speed-up factor of $4/3$. Through experimental evaluation, we showed that the multi-rate model improves the schedulability performance of state-of-the-art.

Chapter 5

Efficient Partitioning Strategy for MC Systems

In this chapter, we present a novel partitioning strategy, called *Utilization Difference based Partitioning* (UDP), for dual-criticality MC systems scheduled on homogeneous multi-core platforms. In a dual-criticality MC system, upon mode switch, the execution demand of HC tasks is increased. The schedulability performance of several single-core MC scheduling algorithms depends on this additional demand. Under UDP strategy, we distribute this additional demand evenly among all processing cores. By balancing this demand we were able to improve the schedulability performance of single-core MC scheduling algorithm applied on each core. This strategy has been previously published in [94].

We present the limitations of the global MC scheduling algorithms and existing partitioning heuristics and present the motivation for this work in Section 5.1. In Section 5.2, we introduce the system model and the existing single-core MC scheduling algorithm used in our experiments. We propose the UDP strategy for assigning MC tasks to cores in Section 5.3. In Section 5.4 we compare the UDP strategy with existing partitioning strategies and show that UDP outperforms all existing strategies in terms of schedulability performance. We conclude this chapter in Section 5.5.

5.1 Introduction

The multi-rate fluid scheduling model proposed in Chapter 4 for implicit-deadline dual-criticality MC systems has better schedulability performance than state-of-the-art. However, fluid scheduling, a variant of global scheduling, makes an assumption that all the LC tasks in the system are discarded upon mode switch. And all the HC tasks in the system simultaneously require additional processing resource. This is a very pessimistic assumption to make in practice which results in under-utilization of processing resource. On the other hand, partitioned MC scheduling restricts the impact of mode switch to one particular core (refer to Chapter 2, Section 2.2.2 for further details). As a result, only a subset of HC tasks simultaneously demands processing resource and only a subset of LC tasks are dropped upon mode switch. Thus, partitioned MC scheduling has better schedulability compared to global MC scheduling and offers inherent support to a subset of LC tasks after the mode switch.

Based on the above motivation, in this chapter, we focus on the design of the partitioned MC scheduling algorithm. As discussed in Section 1.3, partitioned MC scheduling problem comprises of a) task-to-core mapping and b) scheduling of tasks on each core using single-core MC scheduling algorithm. As mentioned in Section 1.3, the task-to-core mapping problem is known to be NP-Complete, and only approximation algorithms are used to assign tasks to cores. The performance of partitioned scheduling algorithms significantly depends on the type of partitioning strategy. Several partitioning strategies [33–35, 80] have been designed for dual-criticality MC systems as discussed in Section 3.2.2. However, none of these strategies perform well for most single-core MC scheduling algorithms due to their inefficiency in balancing the additional demand of the HC tasks after mode transition. Towards addressing this issue, in this chapter, we propose UDP strategy for dual-criticality MC systems.

In a dual-criticality MC system, each HC task has two utilization values - one for each criticality level. When the system switches to the HC mode, the utilization of the HC tasks is increased to meet its demand. The performance of many single-core MC scheduling algorithms depends on the sum of the difference between these two utilization values (denoted in short as *utilization difference*) across all the HC tasks assigned to a core. A smaller utilization difference on a core generally implies that the MC tasks allocated on that core are more likely to be schedulable because the additional demand of HC tasks when the system switches from LC to HC mode is also small. Therefore, under the UDP strategy, we aim to distribute this utilization difference evenly across all cores.

5.2 System Descriptions and Scheduling Algorithms

The problem of scheduling dual-criticality sporadic MC systems on homogenous multi-core platform with m cores is considered. We consider a sporadic task system τ comprising n tasks. The workload of each MC task τ_i is defined using the standard MC workload model presented in Section 2.2.1. We consider both implicit-deadline (relative deadline of tasks equal to minimum release separation time) and constrained-deadline (relative deadline of tasks no larger than minimum release separation time) task systems for our experiments. The execution semantics of the partitioned MC scheduling algorithm is described in Section 2.2.2. That is, under the partitioned MC scheduling, when an HC task in a core switches to the HC mode only that particular core is switched to the HC mode and only the LC tasks in that core are suspended. In Section 5.2.1, we introduce the single-core MC scheduling algorithms used in this chapter.

5.2.1 Single-core MC Scheduling Algorithms

In this section, we describe the single-core MC scheduling algorithms used for evaluating the UDP strategy in our experiments. We consider three different single-core MC algorithms: Adaptive Mixed-Criticality (AMC) [71], Earliest Deadline First with Virtual Deadlines (EDF-VD) [62] and Earliest Carry-over Deadline First (ECDF) [63]. As described in Section 3.1, AMC is a fixed-priority scheduling algorithm with response time analysis based schedulability test. The EDF-VD is a dynamic-priority scheduling algorithm with utilization based schedulability test. The ECDF is also a dynamic-priority scheduling algorithm with demand bound function based schedulability test. A brief description of these algorithms is presented in Section 3.1.

The above choice of algorithms is motivated by the following facts: 1. It enables us to show that the UDP strategy works well with both fixed- and dynamic-priority algorithms, and 2. The pessimism in the schedulability test of these algorithms can be reduced by minimizing the *utilization difference* on a core. Thus, these algorithms are well suited for this study.

5.3 Utilization Difference based Partitioning (UDP)

In this section, we present our utilization difference based partitioning strategy for assigning dual-criticality tasks to cores. We present two strategies based on evenly distributing the utilization difference among all cores; criticality-aware UDP (CA-UDP) strategy and criticality-

unaware UDP (CU-UDP) strategy. The principle behind both these strategies is the same, that is to reduce the maximum difference between total HC utilization and total LC utilization of HC tasks allocated on each core.

Let ϕ_k denote the k^{th} processing core, then the intent is to reduce $\max\{U_H^H(\phi_k) - U_H^L(\phi_k)|1 \leq k \leq m\}$. Suppose that, at any point during the partitioning strategy, we use $\tau(\phi_k)$ to denote the set of tasks assigned to processor ϕ_k until that point. Then, we define system-level utilizations for each core ϕ_k as follows:

$$\begin{aligned} U_L^L(\phi_k) &\stackrel{\text{def}}{=} \sum_{\tau_i \in \tau(\phi_k) \wedge \chi_i = LC} u_i^L \\ U_H^L(\phi_k) &\stackrel{\text{def}}{=} \sum_{\tau_i \in \tau(\phi_k) \wedge \chi_i = HC} u_i^L \\ U_H^H(\phi_k) &\stackrel{\text{def}}{=} \sum_{\tau_i \in \tau(\phi_k) \wedge \chi_i = HC} u_i^H \end{aligned}$$

5.3.1 CA-UDP Strategy

Under CA-UDP strategy, all HC tasks (τ_H) are assigned to cores before assigning any of the LC tasks. Before assigning any tasks to the core, the tasks are sorted based on their utilization values at their respective criticality levels. That is, the HC tasks are sorted based on u_i^H and the LC tasks are sorted based on u_i^L . For the HC tasks, CA-UDP uses worst-fit allocation strategy based on the parameter $U_H^H(\phi_k) - U_H^L(\phi_k)$ *i.e.*, the cores are considered in increasing order of $U_H^H(\phi_k) - U_H^L(\phi_k)$. This ensures that the utilization difference among all cores is evenly distributed. Then, for the LC tasks, CA-UDP uses first-fit allocation strategy *i.e.*, the cores are considered in sequential order from the first core. We evaluate the feasibility of any task allocation using the schedulability test of the single-core MC algorithm. We use any one of the scheduling algorithm presented in Section 5.2.1. If the schedulability test on a core passes, then the task is successfully assigned to that core. If the test fails on a core, then the next core is considered for allocation based on the fitting order. The algorithm returns partitioning failure if the test fails on all cores. The pseudocode for the CA-UDP strategy is presented in Algorithm 2. With the help of an example (Example 7), we illustrate the benefit of CA-UDP by comparing it with another strategy that uses criticality-aware partitioning. The considered strategy uses worst-fit allocation based on u_i^H for the HC tasks and identical strategy as CA-UDP for the LC tasks. We denote this strategy as CA-Wu-F. This strategy is shown to be the good performing partitioning strategy for dual-criticality MC systems [34].

Algorithm 2 CA-UDP**Input:** τ, m **Output:** Set of tasks assigned to core ϕ_k denoted as $\tau(\phi_k)$, where $k \in [1, \dots, m]$

```

  ▷ Partitioning  $\tau_H$ 
  1: Sort  $\tau_H$  in decreasing order of  $u_i^H$ .
  2: for  $i := 1$  to  $\text{length}(\tau_H)$  do
  3:   Sort  $\phi_k$  in increasing order of  $U_H^H(\phi_k) - U_H^L(\phi_k)$ .
  4:   for  $k := 1$  to  $m$  do
  5:     if  $\tau(\phi_k) \cup \tau_i$  is schedulable then
  6:        $\tau(\phi_k) = \tau(\phi_k) \cup \tau_i$  and break.
  7:     end if
  8:   end for
  9:   if  $\tau_i$  could not be allocated then
  10:    Return partitioning failed.
  11:  end if
  12: end for
  ▷ Partitioning  $\tau_L$ 
  13: Sort  $\tau_L$  in decreasing order of  $u_i^L$ .
  14: for  $i := 1$  to  $\text{length}(\tau_L)$  do
  15:   for  $k := 1$  to  $m$  do
  16:     if  $\tau(\phi_k) \cup \tau_i$  is schedulable then
  17:        $\tau(\phi_k) = \tau(\phi_k) \cup \tau_i$  and break.
  18:     end if
  19:   end for
  20:   if  $\tau_i$  could not be allocated then
  21:    Return partitioning failed.
  22:  end if
  23: end for
  24: Return  $\tau(\phi_k), \forall k \in [1, \dots, m]$ .

```

Example 7 Consider an example task system as shown in Table 4.1 to be scheduled on dual-core (ϕ_1 and ϕ_2) platform using the EDF-VD algorithm. We compare CA-UDP strategy with CA-Wu-F partitioning strategy. The task allocation under both these strategies, when scheduled using EDF-VD, is shown in Figure 5.1.

Under CA-Wu-F, task τ_1 is assigned to ϕ_1 and tasks τ_2 and τ_3 are assigned to ϕ_2 . Although task τ_4 can fit on either ϕ_1 or ϕ_2 based on its utilization, it fails to get allocated on any of the cores due to the EDF-VD schedulability test. From EDF-VD schedulability test (Equation (2.3)), we have $U_L^L \leq (1 - U_H^H)/(1 - (U_H^H - U_H^L))$. Then, applying EDF-VD on each core,

$$\begin{aligned}
 U_L^L(\phi_1) &\leq (1 - 0.7)/(1 - (0.7 - 0.5)) & U_L^L(\phi_2) &\leq (1 - 0.9)/(1 - (0.9 - 0.5)) \\
 \Leftrightarrow U_L^L(\phi_1) &\leq 0.375 & \Leftrightarrow U_L^L(\phi_2) &\leq 0.166
 \end{aligned}$$

Since $u_4^L = 0.4$, it cannot be allocated to either of the cores.

Whereas, under CA-UDP, tasks τ_1 and τ_3 are assigned to ϕ_1 and task τ_2 is assigned to ϕ_2 to balance the utilization difference. Then, applying EDF-VD on each core we have,

$$\begin{aligned} U_L^L(\phi_1) &\leq (1 - 1)/(1 - (1 - 0.7)) & U_L^L(\phi_2) &\leq (1 - 0.6)/(1 - (0.6 - 0.3)) \\ \Leftrightarrow U_L^L(\phi_1) &\leq 0 & \Leftrightarrow U_L^L(\phi_2) &\leq 0.571 \end{aligned}$$

Then, task τ_4 can be assigned successfully to ϕ_2 . Thus, by balancing the utilization difference among all cores, we are able to assign more LC tasks and thereby improve schedulability.

Table 5.1: Example task system scheduled on 2 cores using CA-UDP

Task	T_i	D_i	χ_i	u_i^L	u_i^H
τ_1	10	10	HC	0.5	0.7
τ_2	10	10	HC	0.3	0.6
τ_3	10	10	HC	0.2	0.3
τ_4	10	10	LC	0.4	-

Figure 5.1: Comparison of CA-UDP and CA-Wu-F

One of the limitations with criticality-aware partitioning strategies is that the LC tasks with very high utilization values often fail to get assigned to a core because they are considered for allocation only after all the HC tasks have been allocated. To address this problem, we consider the criticality-unaware partitioning strategy in which all tasks with different criticality are treated alike.

5.3.2 CU-UDP Strategy

Algorithm 3 presents the detailed pseudocode for criticality-unaware UDP (CU-UDP) strategy. Under CU-UDP, all tasks (both HC and LC) are sorted in decreasing order of their utilization

Algorithm 3 CU-UDP

Input: τ, m

Output: Set of tasks assigned to core ϕ_k denoted as $\tau(\phi_k)$, where $k \in [1, \dots, m]$

```

  ▷ Partitioning  $\tau$ 
  1: Sort  $\tau$  in decreasing order of utilization ( $u_i^H$  if  $\chi_i = HC$  and  $u_i^L$  if  $\chi_i = LC$ )
  2: for  $i := 1$  to  $\text{length}(\tau)$  do
  3:   if  $\chi_i == HC$  then
  4:     Sort  $\phi_k$  in increasing order of  $U_H^H(\phi_k) - U_H^L(\phi_k)$ 
  5:     for  $k := 1$  to  $m$  do
  6:       if  $\tau(\phi_k) \cup \tau_i$  is schedulable then
  7:          $\tau(\phi_k) = \tau(\phi_k) \cup \tau_i$  and break.
  8:       end if
  9:     end for
  10:    if  $\tau_i$  could not be allocated then
  11:      Return partitioning failed.
  12:    end if
  13:  else
  14:    for  $k := 1$  to  $m$  do
  15:      if  $\tau(\phi_k) \cup \tau_i$  is schedulable then
  16:         $\tau(\phi_k) = \tau(\phi_k) \cup \tau_i$  and break.
  17:      end if
  18:    end for
  19:    if  $\tau_i$  could not be allocated then
  20:      Return partitioning failed.
  21:    end if
  22:  end if
  23: end for
  24: return  $\tau(\phi_k), \forall k \in [1, \dots, m]$ 

```

values at their respective criticality levels *i.e.*, all tasks in τ_H are sorted in decreasing order of their u_i^H and all tasks in τ_L are sorted in decreasing order of their u_i^L . Then, similar to the CA-UDP, all the HC tasks are assigned to cores using worst-fit allocation strategy based on the parameter $U_H^H(\phi_k) - U_H^L(\phi_k)$ and all the LC tasks are assigned to cores using first fit-allocation strategy. The task is said to be successfully assigned to a core if it passes the schedulability test applied to that core. At any point during the partitioning if a task fails the schedulability test on all the cores, then the algorithm returns partitioning failure. To illustrate the benefit of CU-UDP, we compare it with CA-UDP strategy in Example 8.

Example 8 Table 5.2 shows an example task system to be scheduled on the dual-core platform using EDF-VD algorithm. The task allocation under CA-UDP and CU-UDP strategies, when scheduled using EDF-VD, is shown in Figure 5.2. Under CA-UDP, tasks τ_1 and τ_3 are allocated to ϕ_1 and tasks τ_2 and τ_4 are allocated to ϕ_2 . As a result, the EDF-VD schedulability test i.e., $U_L^L \leq (1 - U_H^H)/(1 - (U_H^H - U_H^L))$, fails on both cores when allocating τ_5 . In the case of CU-UDP, tasks τ_1 and τ_2 are allocated to ϕ_1 and ϕ_2 respectively. Also, due to criticality-unaware partitioning, task τ_5 is allocated to ϕ_1 before tasks τ_3 and τ_4 . Then, tasks τ_3 and τ_4 are successfully allocated to ϕ_2 . In this example, CU-UDP was able to prioritize allocation for a heavy utilization LC task τ_5 while still balancing the utilization difference and hence gave a successful partition.

Table 5.2: Example task system scheduled on 2 cores using CU-UDP

Task	T_i	D_i	χ_i	u_i^L	u_i^H
τ_1	10	10	HC	0.3	0.5
τ_2	10	10	HC	0.3	0.5
τ_3	10	10	HC	0.2	0.3
τ_4	10	10	HC	0.2	0.2
τ_5	10	10	LC	0.4	-

Figure 5.2: Comparison of CA-UDP and CU-UDP

5.4 Evaluation

In this section, using the randomly generated task sets we evaluate the UDP strategy using three single-core MC scheduling algorithms as discussed in Section 5.2.1. These algorithms include EDF-VD, ECDF and AMC. We compare the schedulability performance of the UDP algorithms against existing partitioned MC scheduling algorithms. We show that the proposed UDP based algorithms significantly outperform the existing algorithms. Below is the list of existing partitioned algorithms used for comparison in this chapter.

- CA(nosort)-F-F-EDF-VD [33], which is the only partitioned scheduling algorithm with a known speed-up bound. It uses criticality-aware partitioning with no sorting of tasks, first-fit allocation strategy for both HC as well as LC tasks and employs EDF-VD scheduling algorithm.
- ECA-Wu-F-EY [35], is a partitioned algorithm that employs a virtual deadline based single-core MC scheduling algorithm called EY that is identical to EDF-VD and ECDF [64], except that the schedulability test is based on demand bound functions as in ECDF but it is relatively less efficient in terms of schedulability. It uses an enhanced criticality-aware partitioning strategy with worst-fit allocation based on HC mode utilization and first-fit allocation for LC tasks. The enhancement is that it gives preference to heavy utilization LC tasks over HC tasks while the rest of the tasks uses regular criticality-aware partitioning. The tasks are sorted based on the utilization values at their respective criticality levels before being allocated.
- CA-F-F-EY [34], which is also a partitioned algorithm that employs EY scheduling algorithm. It uses a criticality-aware partitioning strategy with a simple first-fit allocation for both HC and LC tasks. The tasks are sorted based on utilization values at their respective criticality levels before being allocated. It has been previously shown that this algorithm dominates all other existing criticality-aware partitioning strategies in experiments.

The first of the above strategies is only applicable for implicit-deadline task systems, while the latter two are applicable for both implicit and constrained-deadline task systems. It is worth noting that there is no existing partitioned MC algorithm that employs a fixed-priority scheduling policy such as AMC. This is mainly because AMC is known to perform poorly when compared to dynamic-priority schemes such as EDF-VD, ECDF and EY.

5.4.1 Task Set Generation

We use the task set generator proposed in [92] to generate the task sets for our experiments. The task set parameters used in our generator are same as those described in Chapter 4, Section 4.9 except for the following:

- $P_H = 0.5$ denotes the percentage of HC tasks in the system.
- T_i , the period of a task τ_i is drawn log-uniformly [93] at random from $[10, 500]$. We ensure that there are at least 50 different possible values for the period.
- The deadline D_i of the task τ_i is drawn uniformly from $[C_i^H, T_i]$ for constrained-deadline task systems.

The total number of tasks in a task system is lower bounded by $m + 1$ and upper bounded by $3 * m$. We generate at least 10,000 task sets for each value of total normalized utilization U_B ($U_B = \max(\frac{U_H^L + U_L^L}{m}, \frac{U_H^H}{m})$). For each such successfully generated task set we evaluate the performance of our partitioning strategies with the existing strategies.

5.4.2 Evaluation of Implicit-deadline Algorithms with known Speed-up Factor

(a) EDF-VD ($m = 2$)

Figure 5.3: Evaluation of implicit-deadline algorithms with known speed-up factor

We compare the overall schedulability performance of UDP strategies with EDF-VD against CA (nosort)-F-F-EDF-VD for implicit-deadline dual-criticality MC systems for varying m and $P_H = 0.5$. We plot the acceptance ratios of the algorithms, *i.e.*, the fraction of task sets deemed to be schedulable versus total normalized utilization U_B in Figure 5.3. As shown, the performance improvement over CA(nosort)-F-F-EDF-VD due to utilization difference based partitioning is as much as 19.9%, 42.8% and 52.9% for $m = 2, 4$ and 8 respectively. Note that, the EDF-VD schedulability test is used in all the three algorithms. Thus, they all have a speed-up factor of $8/3$ as discussed in Section 3.2.2.

5.4.3 Evaluation of Implicit-deadline Algorithms w/o Speed-up Factor

Since ECDF and AMC do not have a known speed-up bound⁴, we compare the performance of these algorithms for different partitioning strategies.

(a) $m = 2$

⁴Although AMC does not have a known speed-up bound, speed-up bounds derived for similar static-priority algorithms [95] can be extended for the proposed UDP strategies with AMC

(b) $m = 4$ (c) $m = 8$

Figure 5.4: Evaluation of implicit-deadline algorithms w/o speed-up factor

We compare the performance of UDP strategies with ECDF and AMC against ECA-Wu-

F-EY and CA-F-F-EY for implicit-deadline dual-criticality MC systems for varying m and $P_H = 0.5$. We plot the acceptance ratios of the algorithms in Figure 5.4. As shown, the performance of CA-UDP and CU-UDP based algorithms perform well as the number of cores increases depicting its scalability, while the performance of the existing algorithms decreases. For implicit-deadline task systems, the performance improvement over existing algorithms is as much as 0.6%, 3.9% and 19.9% under AMC, and 15.5%, 30.9% and 33.1% under ECDF, for $m = 2, 4$ and 8 respectively.

5.4.4 Evaluation of Constrained-deadline Algorithms

We compare the performance of UDP strategies with ECDF and AMC against ECA-Wu-F-EY and CA-F-F-EY for constrained-deadline dual-criticality MC systems. We plot the acceptance ratios of the algorithms for varying m and $P_H = 0.5$ in Figure 5.5. As shown, the performance of the existing algorithms decreases with increasing number of cores, while CA-UDP and CU-UDP based algorithms continue to perform well. For constrained-deadline dual-criticality MC systems, the performance improvement over existing algorithms is as much as 3.3%, 29.1% and 66.3% under AMC, and 22.8%, 44.9% and 86.7% under ECDF, for $m = 2, 4$ and 8 respectively.

(a) $m = 2$

(b) $m = 4$ (c) $m = 8$

Figure 5.5: Evaluation of constrained-deadline algorithms

5.4.5 Performance Evaluation for Varying Task Parameters

(a) Implicit-deadline (EDF-VD)

(b) Constrained-deadline

Figure 5.6: Varying Percentage of Task Criticality

In Figure 5.6 we compare the *weighted acceptance ratio* (given by Equation (4.13)) of the algorithms for varying $P_H \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ and $m \in \{2, 4\}$. For our experiments on varying P_H , we restrict the number of cores to $m \in \{2, 4\}$. This restriction is necessary to ensure that schedulability tests of some of the algorithms (particularly those based on demand bound functions) can be executed in a reasonable amount of time.

Each data point in the plot corresponds to at least 5000 task sets. We compare the performance of UDP strategies with EDF-VD for implicit-deadline task systems and with AMC and ECDF for constrained-deadline dual-criticality MC systems in Figure 5.6(a) and Figure 5.6(b) respectively.

As shown in Figure 5.6(a), the performance of CA-UDP decreases with increasing percentage of task criticality. At very high P_H values, the task set comprises mainly HC tasks and as a result, the LC tasks in the system have high utilization values. Hence, CA-UDP performs poorly. Whereas, the CU-UDP based algorithms perform well irrespective of the P_H values. For constrained-deadline dual-criticality MC systems, the performance of all the strategies with the criticality-aware worst-fit allocation of HC tasks decreases as P_H increases. This is expected because when a large number of HC tasks are present in the system, criticality-aware partitioning behaves like a classical real-time system where worst-fit allocation is known to perform poorly. Similar to the implicit-deadline case, the CU-UDP based algorithms perform well irrespective of the P_H values.

Based on the results from our experiments, we present our findings below:

- Both CA-UDP and CU-UDP based algorithms perform better than all the existing algorithms.
- The performance gap between our algorithms and existing algorithms increases as m increases, depicting its scalability.
- The CA-UDP and CU-UDP strategies with AMC scheduling algorithm perform well compared to EDF-VD and ECDF. Note that, no existing partitioned MC scheduling algorithms employ a fixed-priority scheduling policy such as AMC. This is mainly because AMC is known to perform poorly when compared to dynamic-priority schemes such as ECDF and EY. Our results show that the partitioning strategy can significantly impact the performance of the partitioned MC scheduling algorithm. Since fixed-priority scheduling is generally preferred in safety-critical industries, the improved performance of UDP strategy under AMC is important.

- The schedulability performance of CU-UDP is generally better than CA-UDP irrespective of the schedulability test being used. This is due to prioritizing all the heavy utilization LC tasks over HC tasks while still ensuring balanced utilization difference among all cores.

Thus, we can conclude that the proposed UDP strategies significantly improve the schedulability across a variety of scheduling algorithms. The fact that schedulability under AMC is also better than for strategies using dynamic-priority algorithms, shows that the observed improvements are primarily because of the partitioning strategies, rather than due to the difference in schedulability tests.

5.5 Conclusions

In this chapter, we presented a novel partitioning strategy for assigning dual-criticality MC tasks on homogeneous multi-core platforms. We proposed a utilization difference based partitioning (UDP) scheme for allocating the HC tasks to the cores. Unlike existing partitioning techniques, the UDP strategy utilizes the special feature of MC workload, which is the difference in execution budgets for different criticality levels. The UDP strategy balances this difference in workload across criticality levels on each core.

We presented two variants of UDP strategy - criticality-aware and criticality-unaware partitioning. Through experimental evaluation, we compared the UDP strategies with existing partitioning strategies using different single-core MC scheduling algorithms. The results from our experiments show that the proposed UDP strategies, both criticality-aware and criticality-unaware variants, significantly outperform the existing approaches under both fixed-priority and dynamic-priority scheduling. We also demonstrated that the UDP strategy improves the schedulability performance for both implicit-deadline and constrained-deadline dual-criticality MC systems.

Chapter 6

Semi-Partitioned Mixed-Criticality Scheduling

In this chapter, we introduce the *semi-partitioned mixed-criticality* (SPMC) model for scheduling implicit-deadline dual-criticality MC systems on homogeneous multi-core platforms. In a conventional dual-criticality MC system, upon mode switch, the LC tasks are immediately suspended. In order to support the execution of these LC tasks, under this model, we migrate them to the core that has not experienced mode switch. To guarantee schedulability of these migrating tasks, we reserve the utilization corresponding to these tasks in the core to which they migrate. However, if this core also switches mode then these tasks will be suspended. This model has been previously published in [96].

In Section 6.2, we introduce the system model and the existing EDF-VD scheduling algorithm used in our analysis. We propose the SPMC scheduling model and describe its execution semantics in Section 6.3. In Section 6.4, we derive the schedulability test for the SPMC model and derive the utilization bound of the algorithm. We present a partitioning strategy for the SPMC model in Section 6.5. In Section 6.6, we compare the SPMC model with existing semi-partitioned fixed-priority algorithm and show that SPMC outperforms in terms of schedulability performance and total LC utilization supported. We conclude this chapter in Section 6.7.

6.1 Introduction

The partitioning strategy proposed in Chapter 5 is able to improve the schedulability performance of the single-core MC scheduling algorithm applied on each core. However, when a processing core switches to the HC mode, the LC tasks executing on that processing core are immediately suspended to meet the demand of the HC tasks. This significantly affects the performance of the LC tasks which may not be suitable for many practical systems [37, 50]. Hence, there is a need to improve the service offered to these tasks in the HC mode.

Several techniques have been proposed to improve the service offered to the LC tasks as discussed in Section 3.3. Although the problem was well studied in the context of single-core platforms, for the multi-core case it is least explored. Most techniques developed for multi-core platforms either offer full service to a subset of LC tasks [57, 76] or uniformly degrade all the LC tasks [56, 58]. As discussed in Section 3.3.1, it is better to execute all the LC tasks with their LC execution budget for the longest possible duration. In the context of partitioned MC scheduling, this could be achieved by migrating these LC tasks to the core that has not experienced mode switch. To achieve this, a semi-partitioned MC model has been proposed for dual-criticality MC systems using AMC scheduling algorithm. As discussed in Section 3.3.1, the migrating LC tasks are scheduled with reduced deadlines which affect the overall utilization of cores.

To overcome the abovementioned problem, in this chapter, we propose a reservation based technique to offer full service to the migrating LC tasks when one of the cores switches to the HC mode. In this model, the migrating LC tasks are modeled using servers and the utilization corresponding to these tasks are reserved in the core to which it migrates (denoted as the *destination core*). To keep the predictability of the HC tasks intact, we statically partition them to cores and only allow LC tasks to migrate upon mode switch. The migrated LC tasks are dropped if the destination core experiences a mode switch.

6.2 System Description

In this section, we describe the workload model considered and the system behavior.

We consider a dual-criticality implicit-deadline sporadic task system τ comprising of n tasks scheduled on homogeneous multi-core platform with m processing cores. The workload of each MC task is represented using the MC workload model presented in Section 2.2.1. We assume

that the cost of migrations, cache related delays and scheduling overheads are negligible in our analysis, and included when computing the execution budgets of tasks.

In this work, we assume that the non-migrating tasks are scheduled using EDF-VD scheduling policy and the migrating tasks are scheduled using a server with EDF scheduling policy. For details on EDF-VD scheduling policy and the schedulability test refer to Section 2.3.1.

6.2.1 Execution Semantics

Initially, the system is in the LC mode. In the LC mode, all tasks are statically partitioned to a core - each task executes on a dedicated core. Each core has a set of statically partitioned HC tasks, a set of statically partitioned LC tasks and a set of migrating LC tasks. If any of the HC tasks in a core exceeds its LC execution budget, then that particular core (dedicated core) switches to the HC mode. Thereupon, the non-migrating LC tasks and the HC tasks continue to execute on the same core while the migrating LC tasks are migrated to another core (destination core) that has not experienced mode switch. Since the non-migrating LC tasks are allowed to run in HC mode, we treat them as HC tasks and scale their deadline in the LC mode by x , $D_i^L = x \cdot T_i$. To guarantee the execution of migrating LC tasks, the resources corresponding to these tasks are reserved in the destination core. If both the dedicated and the destination cores switch to the HC mode, the migrating LC tasks common to both these cores are dropped while the migrating LC tasks native to the destination core is migrated to another core.

6.3 Semi-Partitioned Mixed-Criticality (SPMC) Model

In this section, we propose the Semi-Partitioned Mixed-Criticality (SPMC) model to support guaranteed execution of LC tasks in the HC mode. In particular, we assume that all the LC tasks are guaranteed its LC execution budgets when at most one processing core switches to the HC mode. We previously explained the execution semantics of the system (Section 6.2). We now present our reservation based technique to support the migrating LC tasks.

Figure 6.1 shows the scheduling strategy of SPMC. The migrating LC tasks in the system are scheduled using a server. Each server is represented using an execution budget and period. The servers execute at fixed time slots - the start and end of the server period. The online invocation of the server at respective time slot is done with the help of release time and period of the server. We assume that all the servers in the system have the same period. Each core has two

Figure 6.1: SPMC Model

servers corresponding to the migrating LC tasks in the LC mode - incoming migration (IM) and outgoing migration (OM). After the mode switch, both the IM and OM servers are dropped and only the non-migrating LC tasks and HC tasks are active in the core. Upon mode switch, the tasks in the OM server are migrated to the IM server in the destination core.

Before proceeding to the analysis, we define a few parameters and notations used in our analysis. The k^{th} processing core is denoted using ϕ_k . Similar to the system utilization presented in Section 2.2.1, we define the system utilization for each core ϕ_k as follows: $U_L^L(\phi_k)$, $U_H^L(\phi_k)$ and $U_H^H(\phi_k)$. Let $\tau(\phi_k)$ denote the set of tasks assigned to core ϕ_k . The set of non-migrating LC tasks, set of outgoing migrating LC tasks and set of incoming migrating LC tasks are denoted as $\tau_{NM}(\phi_k)$, $\tau_{OM}(\phi_k)$ and $\tau_{IM}(\phi_k)$ respectively. Then, the total LC utilization on each core ϕ_k is represented as,

$$U_L^L(\phi_k) = U_{NM}(\phi_k) + U_{OM}(\phi_k) + U_{IM}(\phi_k)$$

where,

$$U_{NM}(\phi_k) = \sum_{\tau_i \in \tau_{NM}(\phi_k)} u_i^L$$

$$U_{OM}(\phi_k) = \sum_{\tau_i \in \tau_{OM}(\phi_k)} u_i^L$$

$$U_{IM}(\phi_k) = \sum_{\tau_i \in \tau_{IM}(\phi_k)} u_i^L$$

Intuitively, $U_{NM}(\phi_k)$ denotes the total utilization of the LC tasks that execute in both the modes of core ϕ_k . $U_{OM}(\phi_k)$ denotes the total utilization of the LC tasks migrated from core ϕ_k .

Likewise, $U_{IM}(\phi_k)$ denotes the total utilization of the LC tasks migrated to core ϕ_k .

6.3.1 Bounding the Server Utilization

There is an inherent overhead involved in using the server. Several techniques have been proposed in the real-time systems domain for determining the server overhead. The server structure used in our study is similar to the one proposed in [97]. Hence, we extend the analysis presented in [97] to derive an upper bound for the server overhead. Let T_{min} , the smallest period of all tasks, be defined as

$$T_{min} = \min \left(D_i^L \ \forall \tau_i \in \{\tau_H, \tau_{NM}\}, T_j \ \forall \tau_j \in \{\tau_{OM}, \tau_{IM}\} \right) \quad (6.1)$$

Since we use EDF-VD to schedule the non-server tasks (the HC tasks and the non-migrating LC tasks), and they are allowed to run in the HC mode, their deadline in the LC mode is scaled by x , $D_i^L = x \cdot T_i$. Hence, when we account for the minimum period, we use D_i^L for these tasks and T_i for the remaining tasks. The actual timeline is divided into time slots of fixed length, denoted as *server period*, is given by,

$$S = \frac{T_{min}}{\delta} \quad (6.2)$$

where δ denotes the server period inflation factor. A smaller δ value denotes the larger server period and vice versa.

Let the server reserve corresponding to the outgoing migrating tasks and incoming migrating tasks on core ϕ_k be denoted as OM_k and IM_k respectively. Likewise, OM_l and IM_l denote the corresponding reserves on core ϕ_l . Since the server reserves cannot be accommodated in the HC mode, upon mode switch, the LC tasks executing in these servers are either dropped or migrated to another core. Let us suppose, we migrate the tasks in OM_k to IM_l upon mode switch by ϕ_k , and migrate the tasks in OM_l to IM_k upon mode switch by ϕ_l . The server configuration in ϕ_k and ϕ_l are shown in Figure 6.1. As can be seen, the server OM_k is reserved in ϕ_k at the end of server period and the server OM_l is reserved in ϕ_l at the start of server period. The servers IM_k and IM_l are aligned corresponding to the servers OM_l and OM_k respectively to ensure that the migrating LC tasks meet their deadlines after the mode switch. Note that, the servers IM_k and IM_l or the servers OM_l and OM_k all are not aligned either at the start or end of the server period respectively. This is required to ensure that after the mode switch the incoming migrating tasks

from one core (say those from OM_k) can continue to execute (say in IM_l) on another core⁵.

We now upper bound the total utilization that has to be reserved in the server for the server tasks to be schedulable by considering the phasing of tasks and the server boundaries. We derive the bound for one of the server configurations (say core ϕ_k) and the analysis holds for the other (core ϕ_l). The actual reserve allocated for the server tasks must be greater than its utilization for it to be schedulable. Then, for any interval of length, it must hold that

$$\frac{OM_k + IM_k}{S} \geq U_{OM}(\phi_k) + U_{IM}(\phi_k)$$

To determine the lower bound of $\frac{OM_k + IM_k}{S}$, we utilize the below result from [97].

Proposition 5 (Section 3.2 (Observations) in [97]) *Let a and b denote the two non-overlapping reserves on some processing core, S denotes the server period and δ denotes the server period inflation factor. For any two consecutive windows of execution with adjacent reserves a and b merged, the tasks in reserves are separated by $S - a - b$ interval units of idleness i.e., any interval of time containing z complete server reserves, may contain upto $z + 1$ complete server task idleness. Then, the fraction of time available for tasks (whose release separation time is at least $\delta \cdot S$ units) in these periodic reserves between its arrival time and the deadline is lower bounded by,*

$$\min_{\substack{z=\delta \\ z \in \mathbb{N}}}^{\infty} \frac{z \cdot (a + b)}{(z + 1) \cdot S - (a + b)} = \frac{\delta \cdot (a + b)}{(\delta + 1) \cdot S - (a + b)}$$

where z denotes the window number.

Using Proposition 5, the release separation time of the tasks (i.e., $T_i \geq \delta \cdot S$) and periodic availability of the reserve, $\frac{OM_k + IM_k}{S}$ is lower bounded by

$$\min_{\substack{z=\delta \\ z \in \mathbb{N}}}^{\infty} \frac{z \cdot (OM_k + IM_k)}{(z + 1) \cdot S - (OM_k + IM_k)} = \frac{\delta \cdot (OM_k + IM_k)}{(\delta + 1) \cdot S - (OM_k + IM_k)}$$

In order to schedule the server tasks such that no task deadline is missed, the reserve allocated must be sufficiently inflated. Let α denote the required inflation factor for each of the servers to

⁵Note, a migrating LC task is said to be successfully allocated if it can be successfully assigned to a pair of cores. See Section 6.5 for further details. If the number of processing cores is odd, then some of the cores might not have servers either at the start or at the end of the server period.

meet task deadlines irrespective of the phasing between task arrivals and the server boundaries⁶. Then,

$$OM_k = S \cdot (\alpha + U_{OM}(\phi_k)) \quad (6.3)$$

$$IM_k = S \cdot (\alpha + U_{IM}(\phi_k)) \quad (6.4)$$

Therefore, for the server tasks to be schedulable, it must hold that:

$$\begin{aligned} & \frac{\delta \cdot (OM_k + IM_k)}{(\delta + 1) \cdot S - (OM_k + IM_k)} \geq U_{OM}(\phi_k) + U_{IM}(\phi_k) \\ \Leftrightarrow & \frac{\delta \cdot (2\alpha + U_{OM}(\phi_k) + U_{IM}(\phi_k))}{(\delta + 1) - (2\alpha + U_{OM}(\phi_k) + U_{IM}(\phi_k))} \geq U_{OM}(\phi_k) + U_{IM}(\phi_k) \end{aligned}$$

Let $U_s(\phi_k)$ denote the total server utilization on core ϕ_k . Then, we have

$$\begin{aligned} & \frac{\delta \cdot (2\alpha + U_s(\phi_k))}{(\delta + 1) - (2\alpha + U_s(\phi_k))} \geq U_s(\phi_k) \\ \Leftrightarrow & \delta(2\alpha + U_s(\phi_k)) \geq (\delta + 1)U_s(\phi_k) - (2\alpha + U_s(\phi_k))U_s(\phi_k) \\ \Leftrightarrow & 2\alpha\delta + \delta U_s(\phi_k) \geq \delta U_s(\phi_k) + U_s(\phi_k) - 2\alpha U_s(\phi_k) - U_s(\phi_k)^2 \\ \Leftrightarrow & 2\alpha\delta \geq U_s(\phi_k) - 2\alpha U_s(\phi_k) - U_s(\phi_k)^2 \\ \Leftrightarrow & 2\alpha\delta + 2\alpha U_s(\phi_k) \geq U_s(\phi_k) - U_s(\phi_k)^2 \\ \Leftrightarrow & 2\alpha(\delta + U_s(\phi_k)) \geq U_s(\phi_k)(1 - U_s(\phi_k)) \\ \Leftrightarrow & \alpha \geq \frac{U_s(\phi_k)(1 - U_s(\phi_k))}{2(\delta + U_s(\phi_k))} \end{aligned} \quad (6.5)$$

where, $U_s(\phi_k) = U_{OM}(\phi_k) + U_{IM}(\phi_k)$. In order to determine the upper bound on total server utilization, we try to determine the maximum value for the inflation factor α . For $\alpha \geq 0$, α is maximum at $U_s^{max}(\phi_k) = \sqrt{(\delta(\delta + 1))} - \delta$ given by,

$$\alpha(U_s^{max}(\phi_k)) \geq \frac{1}{2}[2\delta - 2\sqrt{\delta(\delta + 1)} + 1] \quad (6.6)$$

Note that, the server reserves OM_k and IM_k on the same core ϕ_k cannot overlap in time. Therefore, for the schedulability of the server tasks, we require

$$(U_{OM}(\phi_k) + \alpha) + (U_{IM}(\phi_k) + \alpha) \leq 1 \Rightarrow U_s(\phi_k) \leq 1 - 2\alpha$$

⁶For servers IM_k and OM_k , the release time is updated as $0, S, 2S, \dots$ and $S - (U_{OM}(\phi_k) + \alpha), 2S - (U_{OM}(\phi_k) + \alpha), \dots$ respectively.

6.4 Schedulability Analysis

In Section 6.3, we derived the total server utilization bound such that all server tasks are schedulable independent of non-server tasks. We also bound the inflation factor for each of the servers. In this section, we will derive the lowest system utilization on each core for which the non-server tasks miss their deadline when the utilization for the servers are reserved.

Let t_f denote the first time instant when a deadline miss occurs and L denote the busy interval (*i.e.*, the core has been busy executing for L time units before t_f). Let t_s denote the total time duration for which the server tasks are active in the interval L . For the non-server tasks to be schedulable, its demand t_{dem} must not be greater than the time available for the non-server tasks (t_{ns}). Then, for a deadline miss to occur,

$$t_{dem} > t_{ns} = L - t_s \quad (6.7)$$

We separately derive the non-server tasks demand and the total utilization bound for each of the criticality modes as the demand of these tasks are different depending on the mode in which they are executing.

6.4.1 Low Criticality Mode

In this section, we will derive the total utilization, inclusive of the non-server and server tasks utilization, that can be supported on core ϕ_k such that no tasks miss their deadlines in the LC mode.

The non-server tasks comprise of both non-migrating LC and HC tasks. Then, the total demand of such tasks in the LC mode is given by Equation (6.8).

$$t_{dem} \leq \sum_{\tau_i \in \tau_{NM}(\phi_k)} \left\lfloor \frac{L}{x \cdot T_i} \right\rfloor \cdot C_i^L + \sum_{\substack{\tau_i \in \tau(\phi_k) \\ \chi_i = HC}} \left\lfloor \frac{L}{x \cdot T_i} \right\rfloor \cdot C_i^L \quad (6.8)$$

Note that,

$$\sum_{\tau_i \in \tau_{NM}(\phi_k)} \left\lfloor \frac{L}{x \cdot T_i} \right\rfloor \cdot C_i^L \leq \sum_{\tau_i \in \tau_{NM}(\phi_k)} \frac{L}{x \cdot T_i} \cdot C_i^L = \frac{L}{x} \cdot U_{NM}(\phi_k)$$

$$\sum_{\substack{\tau_i \in \tau(\phi_k) \\ \chi_i = HC}} \left\lfloor \frac{L}{x \cdot T_i} \right\rfloor \cdot C_i^L \leq \sum_{\substack{\tau_i \in \tau(\phi_k) \\ \chi_i = HC}} \frac{L}{x \cdot T_i} \cdot C_i^L = \frac{L}{x} \cdot U_H^L(\phi_k)$$

Then, combining Equations (6.7) and (6.8) we obtain:

$$\begin{aligned} L \cdot \frac{U_{NM}(\phi_k) + U_H^L(\phi_k)}{x} &> L - t_s \\ \Leftrightarrow \frac{U_{NM}(\phi_k) + U_H^L(\phi_k)}{x} &> \frac{L - t_s}{L} \end{aligned} \quad (6.9)$$

We utilize the result from [97] to solve the above inequality. Proposition 6 captures the lower bound on the right-hand side term of the above inequality.

Proposition 6 (Lemma 1, from [97]) *For any interval of length $L \geq \delta \cdot S$, if t_s denotes the cumulative time (total time spent in both reserves) within said interval belonging to reserves a and b on some processing core, it holds that*

$$\frac{L - t_s}{L} \geq \frac{\delta \cdot (S - a - b)}{\delta \cdot S + (a + b)}$$

For any $L \geq T_i$, where T_i (or D_i^L) denote the inter-arrival time of whichever task misses its deadline, $L \geq \delta \cdot S$ (since $T_i \geq T_{min} = \delta \cdot S$). It then holds that,

$$\frac{L - t_{sp}}{L} \geq \frac{\delta \cdot (S - OM_k - IM_k)}{\delta \cdot S + (OM_k + IM_k)} \quad (6.10)$$

using Proposition 6 and replacing reserve a with server OM_k and reserve b with IM_k . For any two consecutive windows of execution, corresponding to all server reserves merged, the server tasks are separated by server task idleness of $S - OM_k - IM_k$ units. Thus, for any interval of time containing z full server task idleness, there can be up to $z + 1$ complete server reserves. Then, by combining Equations (6.9) and (6.10), we get

$$\frac{\delta \cdot (S - OM_k - IM_k)}{\delta \cdot S + (OM_k + IM_k)} < \frac{U_{NM}(\phi_k) + U_H^L(\phi_k)}{x} \quad (6.11)$$

Adding $U_s(\phi_k)$ on both sides and rewriting using equations (6.3) and (6.4) yields:

$$U_s(\phi_k) + \frac{U_{NM}(\phi_k) + U_H^L(\phi_k)}{x} > U_s(\phi_k) + \frac{\delta \cdot S - \delta \cdot S \cdot (U_s(\phi_k) + 2\alpha)}{S(\delta + U_s(\phi_k) + 2\alpha)}$$

Further rewriting yields:

$$U_s(\phi_k) + \frac{U_{NM}(\phi_k) + U_H^L(\phi_k)}{x} > U_s(\phi_k) + \frac{\delta \cdot (1 - U_s(\phi_k) - 2\alpha)}{\delta + U_s(\phi_k) + 2\alpha} \quad (6.12)$$

Let U_B^L denote the minimum of the right-hand side in Equation (6.12). Then, no non-server task misses its deadline in the LC mode as long as the cumulative utilization of both the server and the non-server tasks on any core does not exceed U_B^L . Note that, U_B^L is a function of $U_s(\phi_k)$. In order to compute U_B^L , we substitute Equation (6.6) in Equation (6.12) and determine the lower bound on the right hand side of Equation (6.12). Substituting $y = U_s(\phi_k) + \frac{U_{NM}(\phi_k) + U_H^L(\phi_k)}{x}$ and $x = U_s(\phi_k)$ in Equation (6.12), we have,

$$y > x + \frac{\delta \cdot (1 - x - 2\alpha)}{\delta + x + 2\alpha}$$

Then, we determine the minima using derivatives

$$\begin{aligned} \frac{dy}{dx} &= 1 + \frac{(\delta + x + 2\alpha)(-\delta) - \delta(1 - x - 2\alpha)(1)}{(\delta + x + 2\alpha)^2} \\ &1 + \frac{(\delta + x + 2\alpha)(-\delta) - \delta(1 - x - 2\alpha)(1)}{(\delta + x + 2\alpha)^2} = 0 \\ &\Leftrightarrow (\delta + x + 2\alpha)^2 + (\delta + x + 2\alpha)(-\delta) - \delta(1 - x - 2\alpha)(1) = 0 \\ &\Leftrightarrow (\delta + x + 2\alpha)^2 + (-\delta^2 - x\delta - 2\alpha\delta) - (\delta - x\delta - 2\alpha\delta) = 0 \\ &\Leftrightarrow (\delta^2 + x^2 + 4\alpha^2 + 2x\delta + 4x\alpha + 4\alpha\delta) + (-\delta^2 - \delta) = 0 \\ &\Leftrightarrow x^2 + (2\delta + 4\alpha)x + ((\delta + 2\alpha)^2 - \delta^2 - \delta) = 0 \end{aligned}$$

Since x is non-negative, we determine the positive roots of the above equation,

$$\begin{aligned} x &= \frac{-(2\delta + 4\alpha) + \sqrt{(2\delta + 4\alpha)^2 - 4((\delta + 2\alpha)^2 - \delta^2 - \delta)}}{2} \\ &= -(\delta + 2\alpha) + \sqrt{(\delta + 2\alpha)^2 - ((\delta + 2\alpha)^2 - \delta^2 - \delta)} \\ &= -(\delta + 2\alpha) + \sqrt{\delta^2 + \delta} \\ &= -(\delta + 2\alpha) + \frac{1}{2}[2\delta - 2\sqrt{\delta(\delta + 1)} + 1] + \sqrt{\delta^2 + \delta} \quad (\text{from Equation (6.6)}) \\ &= -(3\delta - 2\sqrt{\delta(\delta + 1)} + 1) + \sqrt{\delta^2 + \delta} \\ &= 3\sqrt{\delta(\delta + 1)} - 3\delta + 1 \end{aligned}$$

Thus, Equation (6.12) is minimized for

$$U_s(\phi_k) = 3(\sqrt{\delta(\delta+1)} - \delta) - 1 \quad (6.13)$$

and U_B^L is given by substituting $U_s(\phi_k)$ and α in from Equation (6.12).

$$\begin{aligned} & 3(\sqrt{\delta(\delta+1)} - \delta) - 1 + \frac{\delta \cdot (1 - (3(\sqrt{\delta(\delta+1)} - \delta) - 1) - 2\frac{1}{2}[2\delta - 2\sqrt{\delta(\delta+1)} + 1])}{\delta + 3(\sqrt{\delta(\delta+1)} - \delta) - 1 + 2\frac{1}{2}[2\delta - 2\sqrt{\delta(\delta+1)} + 1]} \\ \Leftrightarrow & 3\sqrt{\delta(\delta+1)} - 3\delta - 1 + \frac{\delta \cdot (1 - 3\sqrt{\delta(\delta+1)} + 3\delta + 1 - 2\delta + 2\sqrt{\delta(\delta+1)} - 1)}{\delta + 3\sqrt{\delta(\delta+1)} - 3\delta - 1 + 2\delta - 2\sqrt{\delta(\delta+1)} + 1} \\ \Leftrightarrow & 3\sqrt{\delta(\delta+1)} - 3\delta - 1 + \frac{\delta \cdot (1 - \sqrt{\delta(\delta+1)} + \delta)}{\sqrt{\delta(\delta+1)}} \\ \Leftrightarrow & 3\sqrt{\delta(\delta+1)} - 3\delta - 1 + \frac{(\delta(\delta+1)) - \delta\sqrt{\delta(\delta+1)}}{\sqrt{\delta(\delta+1)}} \\ \Leftrightarrow & 3\sqrt{\delta(\delta+1)} - 3\delta - 1 + \sqrt{\delta(\delta+1)} - \delta \\ \Leftrightarrow & 4\sqrt{\delta(\delta+1)} - 4\delta - 1 \\ \Leftrightarrow & 4\frac{1}{2}[2\sqrt{\delta(\delta+1)} - 2\delta - 1] + 1 \end{aligned}$$

which is $1 - 4\alpha$.

Thus, for all the tasks to be schedulable in the LC mode the total utilization of the server and the non-server tasks should be within U_B^L .

$$U_{OM}(\phi_k) + U_{IM}(\phi_k) + \frac{U_{NM}(\phi_k) + U_H^L(\phi_k)}{x} \leq U_B^L = 1 - 4\alpha \quad (6.14)$$

For $\delta = 1$, the LC mode utilization bound is given by $U_B^L = 0.6568$. As δ increases, the LC utilization bound increases and so does the number of preemptions since the server period $S = T_{min}/\delta$.

6.4.2 High Criticality Mode

We have derived the utilization bound of the system in the LC mode. In this section, we derive such utilization bound for the HC mode. When a core (ϕ_k) switches to the HC mode, conventional MC scheduling immediately discards all the LC tasks. In the case of SPMC, the HC tasks and the non-migrating LC tasks continue to run after the mode switch, while the servers corresponding to the migrating LC tasks (IM_k and OM_k) are dropped upon mode switch and

the task in OM_k are migrated to another core.

Suppose that t_f denotes the time instant when the first deadline miss occurs in HC mode and t_m denotes the time instant when the mode switch to HC occurs, where $t_m < t_f$. Assume that Φ is the minimal set of jobs released by $\tau(\phi_k)$ which leads to a deadline miss at t_f . By minimality we mean that the schedulability of the subset of Φ is guaranteed. Let η_i denote the demand of task τ_i in the interval $[0, t_f]$. Let J_1 denote the job with the earliest release time among all the jobs in Φ that execute in $[t_m, t_f]$. Let a_1 and d_1 denote its release time and deadline respectively.

Since we used EDF-VD to schedule the tasks, we use some of the existing results of EDF-VD from literature to derive the schedulability conditions in the HC mode.

Proposition 7 (Fact 1, from [62]) *All jobs in Φ that execute in $[t_m, t_f]$ have deadline $\leq t_f$.*

Only the jobs with deadline $\leq t_f$ are possible to cause a deadline miss at t_f . If a job with deadline $> t_f$ executes and is still in Φ , it will contradict the assumed minimality of Φ .

Proposition 8 (from [62]) *The mode switch time instant t_m holds that*

$$t_m < (a_1 + x(t_f - a_1))$$

The job J_1 has a virtual deadline at $a_1 + x(d_1 - a_1)$. Since it executes in the interval $[t_m, t_f]$, its deadline d_1 should be $< t_f$ and its virtual deadline should be $> t_m$. Otherwise, it would have completed its LC execution by its virtual deadline.

Proposition 9 (Fact 2, from [62]) *Any LC task τ_i that is dropped at mode switch instant t_m has*

$$\eta_i \leq u_i^L(a_1 + x(t_f - a_1))$$

Proposition 9 is used to bound the cumulative execution of the migrating LC tasks that are discarded upon the mode switch. For the migrating LC tasks to execute after a_1 , its deadline should be $\leq (a_1 + x(d_1 - a_1))$, the virtual deadline of job J_1 . So, no tasks with deadline $> (a_1 + x(d_1 - a_1))$ will execute.

Proposition 10 (Fact 3, from [62]) *Any HC task τ_i has*

$$\eta_i \leq \frac{u_i^L}{x} a_1 + (t_f - a_1) u_i^H$$

Proposition 10 is used to bound the demand of the HC tasks. The result stems from the fact that these tasks have different execution requirement and deadline in the LC mode and HC mode.

Lemma 6 *For any LC task τ_i that is allowed to run beyond mode switch instant t_m , it holds that*

$$\eta_i \leq u_i^L(a_1 + (t_f - a_1))$$

Proof: Only the non-migrating LC jobs execute after t_m . There are two cases based on τ_i 's job release.

Case: If τ_i releases a job at or after a_1 . It is trivial to see this case holds. Let $a_i \geq a_1$ denote the first release. Then, the cumulative execution of τ_i is at most $a_i \cdot u_i^L + (t_f - a_i)u_i^L$.

Case: If τ_i does not release a job at or after a_1 . Let a_i denote the release time $< a_1$ and d_i denotes its deadline. We have $\eta_i \leq a_i \cdot u_i^L + (d_i - a_i)u_i^L \leq d_i \cdot u_i^L$. Each job of task τ_i must have a deadline $d_i \leq (a_1 + x(t_f - a_1))$. Therefore, its cumulative execution is at most $u_i^L t_f$. ■

With Propositions 9-10 and Lemma 6, we can derive the schedulability condition for the SPMC model in the HC mode. Let us sum the cumulative execution demand of both the server and the non-server tasks over $[0, t_f]$.

$$\begin{aligned} & \sum_{\substack{\tau_i \in \tau(\phi_k) \\ \chi_i = LC}} \eta_i + \sum_{\substack{\tau_i \in \tau(\phi_k) \\ \chi_i = HC}} \eta_i \Leftrightarrow \sum_{\tau_{NM}(\phi_k)} \eta_i + \sum_{\tau_{OM}(\phi_k)} \eta_i + \sum_{\tau_{IM}(\phi_k)} \eta_i + \sum_{\substack{\tau_i \in \tau(\phi_k) \\ \chi_i = HC}} \eta_i \\ & \leq \sum_{\tau_i \in \tau_{NM}(\phi_k)} \left(\frac{a_1}{x} + (t_f - a_1) \right) u_i^L + (a_1 + x(t_f - a_1))(\alpha + U_{OM}(\phi_k)) \\ & \quad + (a_1 + x(t_f - a_1))(\alpha + U_{IM}(\phi_k)) + \sum_{\substack{\tau_i \in \tau(\phi_k) \\ \chi_i = HC}} \frac{u_i^L}{x} a_1 + (t_f - a_1) u_i^H \\ & \quad \text{(Using Propositions 9-10, Lemma 6 and } \frac{a_1}{x} \geq a_1) \\ & \Leftrightarrow a_1 \left(U_{OM}(\phi_k) + U_{IM}(\phi_k) + 2\alpha + \frac{U_{NM}(\phi_k) + U_H^L(\phi_k)}{x} \right) \\ & \quad + (t_f - a_1) \left(x(U_{OM}(\phi_k) + U_{IM}(\phi_k) + 2\alpha) \right) + (t_f - a_1) \left(U_{NM}(\phi_k) + U_H^H(\phi_k) \right) \\ & \leq a_1 \left(1 - 2\alpha \right) + (t_f - a_1) \left(x(U_{OM}(\phi_k) + U_{IM}(\phi_k) + 2\alpha) \right) \\ & \quad + (t_f - a_1) \left(U_{NM}(\phi_k) + U_H^H(\phi_k) \right) \quad \text{(Using Equation (6.14))} \end{aligned}$$

Since the instance Φ is infeasible, we obtain

$$\begin{aligned} & a_1(1 - 2\alpha) + (t_f - a_1) \left(x(U_{OM}(\phi_k) + U_{IM}(\phi_k) + 2\alpha) \right) \\ & \quad + (t_f - a_1) \left(U_{NM}(\phi_k) + U_H^H(\phi_k) \right) > t_f \\ \Leftrightarrow & x(U_{OM}(\phi_k) + U_{IM}(\phi_k) + 2\alpha) + U_{NM}(\phi_k) + U_H^H(\phi_k) > (t_f - a_1(1 - 2\alpha))/(t_f - a_1) \end{aligned}$$

By taking the contrapositive, we derive the schedulability condition for the SPMC in the HC mode.

$$x(U_{OM}(\phi_k) + U_{IM}(\phi_k)) + U_{NM}(\phi_k) + U_H^H(\phi_k) \leq (t_f - a_1(1 - 2\alpha))/(t_f - a_1) - 2\alpha x$$

Since $1 - 2\alpha \leq 1$, we have

$$x(U_{OM}(\phi_k) + U_{IM}(\phi_k)) + U_{NM}(\phi_k) + U_H^H(\phi_k) \leq 1 - 2\alpha x \quad (6.15)$$

Thus, for the tasks to be schedulable in the HC mode, the cumulative utilization of the server and the non-server tasks should be within U_B^H , where $U_B^H = 1 - 2\alpha x$.

6.4.3 Bounding the number of preemptions

In this section, we try to bound the number of preemption that may arise in the SPMC scheduling. Since we utilize servers to schedule the tasks and they are invoked periodically, frequent preemptions are bound to happen. In reality, the preemption cost is not negligible and can significantly affect the performance of the system when more preemptions are generated. Therefore, it is essential for a scheduling algorithm to have fewer preemptions.

Under SPMC scheduling, preemptions occur only during task arrivals and server boundaries. We now present the upper bound for the number of preemptions within a time window. Let $N_{\phi_k}(t)$ denote the number of jobs that may occur in time interval t by both the server and non-server tasks assigned to core ϕ_k . Then, the algorithm generates at most $3 \cdot \delta \cdot \lceil \frac{t_l}{T_{min}} \rceil + N_{\phi_k}(t)$, where t_l is the fraction of the time window in the LC mode. In the LC mode, every time slot $[z \cdot S, (z + 1) \cdot S]$ contains three instants corresponding to the server boundaries and in the HC mode, there are no such instants as the servers are dropped. The number of time slots that the time window of length t_l can overlap with is upper bounded by $\lceil \frac{t_l \cdot \delta}{T_{min}} \rceil \leq \delta \cdot \lceil \frac{t_l}{T_{min}} \rceil$. Thus, the

total number of preemptions due to the servers is upper bound by $3 \cdot \delta \cdot \lceil \frac{t_l}{T_{min}} \rceil$.

It can be observed that the preemption bound is a function of δ . From Equation (6.2), we have $\delta = T_{min}/S$. Thus, a larger server period generates fewer preemptions while a smaller server period generates too many preemptions. Although a higher utilization bound can be achieved with a larger δ value (smaller server period), it results in huge amount of preemptions which may not be suitable in practice. Thus, there is a need to determine an ideal δ value such that the utilization bound is maximized while preemption count is maintained within a reasonable limit. In the classical real-time systems case, $\delta \leq 4$ is considered to be the reasonable value of δ to maintain a low number of preemptions [98].

6.5 Task Allocation Strategy for SPMC

In Section 6.4, we derived the schedulability test for the SPMC model. In this section, we will describe how to assign tasks to the multi-core platform under SPMC. As described in Chapter 5, the task assignment strategy significantly affects the performance of the schedulability test applied on each core.

Under the SPMC scheduling, the HC tasks are statically partitioned to a core and do not migrate at runtime. Only the LC tasks are allowed to migrate at runtime depending on the execution mode. Hence, we utilize a criticality-aware partitioning strategy where the HC tasks are prioritized over LC tasks for assignment. Since SPMC involves more preemptions than general partitioned scheduling, we first try the non-migration approach. Under the non-migration approach, the tasks are statically assigned based on their conservative execution budget (worst-case reservation) using decreasing utilization first-fit allocation policy. The semi-partitioned approach will only be applied if the non-migration approach fails. The detailed allocation strategy of the SPMC algorithm is as follows:

1. *Criticality-aware decreasing utilization.* The task set is first sorted based on their criticality and then based on their respective utilization, u_i^H if $\tau_i \in \tau_H$ and u_i^L if $\tau_i \in \tau_L$.
2. *Heavy task allocation.* All the heavy tasks, those with utilization greater than the utilization bound, are assigned to a dedicated core. If the number of heavy tasks is greater than the number of cores then the task system will be deemed unschedulable.
3. *HC task allocation.* Then, all the HC tasks are assigned to a dedicated core using UDP

strategy (refer to Chapter 5 for more details). To enable reservation later, the HC tasks are assigned to a core until the utilization bounds (Equations (6.14) and (6.15)) on that core are not violated. It has been shown in Section 5.4 that UDP strategy improves the performance of the EDF-VD algorithm. Since SPMC internally uses EDF-VD to schedule the tasks, UDP strategy helps in improving the schedulability performance.

4. *HC task schedulability.* Once all cores have been checked and if neither of them can schedule the HC task, then the task system will be deemed unschedulable.
5. *LC task allocation.* When all the HC tasks are assigned to cores, the LC tasks are considered for the assignment using UDP allocation.
6. *Non-migrating LC task allocation.* An LC task is assigned to a dedicated core (as non-migrating task) if it can execute in the HC mode on the same core such that the utilization bound conditions (Equations (6.14) and (6.15)) are not violated. Otherwise, the task will be marked as migrating LC task. Note that, the non-migrating LC tasks are treated like HC tasks.
7. *Migrating LC task allocation.* Then, the migrating LC tasks will be considered for the assignment using worst-fit allocation based on total utilization in LC mode ($U_L^L(\phi_k) + U_H^L(\phi_k)$). For LC tasks that are dropped upon mode switch, this strategy has been known to perform well.
8. *Reserve allocation for Migrating LC task.* The migrating LC task will be allocated to a pair of cores - one dedicated core in the LC mode and another core for migration if the dedicated core switches to the HC mode. The migrating LC task is said to be allocated only if it is schedulable in both the cores.
9. *LC task schedulability.* If the migrating LC task cannot be allocated to any of the cores, then the task system is said to be unschedulable.

6.6 Evaluation

In this section, we use randomly generated synthetic task sets to evaluate the performance of the SPMC algorithm. We compare it with existing algorithms that can offer full service to the LC tasks after the mode switch by one of the processing cores. Through comparison, we show

that SPMC has better schedulability performance and offer higher LC utilization support than those algorithms. These algorithms include Semi-FP [88] and WCR-EDF.

- **Semi-FP** is a semi-partitioned algorithm based on fixed-priority scheduling for dual-core MC systems. At the mode switch, the algorithm migrates the LC tasks from one core to another. When the second core switches to the HC mode, all the LC tasks in that core will be discarded. It uses criticality-aware partitioning with worst-fit allocation based on the utilization values at their respective criticality levels.
- **WCR-EDF** is a partitioned algorithm based on EDF scheduling. The tasks are provisioned by their conservative execution budget. Due to this worst-case reservation strategy, the LC tasks in all the cores continue to execute with their LC execution budget after the mode switch. It uses criticality-aware partitioning with the first-fit allocation.

We evaluate the performance of SPMC algorithm for two different server periods: $\delta = 1$ and $\delta = 4$. The largest server period *i.e.*, $\delta = 1$, has the least performance and lowest preemption count while $\delta = 4$ is considered to be the server period with reasonable preemption count as mentioned in Section 6.4.3.

6.6.1 Task Set Generation

We use the task set generator proposed in [92] to generate the task sets for our experiments. The task parameters used in our generator are same as those described in Chapter 4, Section 4.9 except for the following:

- $P_L \in [0.1, 0.2, \dots, 0.9]$ denotes the percentage of LC tasks in the system.

The total number of tasks in the system is lower bounded by $m + 1$ and upper bounded by $10 * m$. We generate at least 10,000 task sets for each value of total normalized utilization U_B ($U_B = \max(\frac{U_H^L + U_L^L}{m}, \frac{U_H^H}{m})$). For each such successfully generated task set we evaluate the performance of our scheduling algorithm with the existing algorithms.

6.6.2 Evaluation of Schedulability Performance

We compare the schedulability performance of SPMC against Semi-FP for $m = 2$ and $P_L = 0.5$ in Figure 6.2. We plot the acceptance ratio (*i.e.*, the fraction of schedulable task sets)

Figure 6.2: Evaluation of Schedulability Performance

of the algorithms versus total normalized utilization U_B . We only plot the results for $U_B \geq 0.5$ because all the algorithms have performance close to 1 otherwise. As seen, $\text{SPMC}(\delta = 1)$ and $\text{SPMC}(\delta = 4)$ outperforms SEMI-FP by as much as 12.7% and 24.3% respectively for $U_B = 0.8$. In case of WCR-EDF , the performance of $\text{SPMC}(\delta = 1)$ seems less significant compared to WCR-EDF because the allocation strategy (presented in Section 6.5) tries to assign LC tasks as non-migrating (static partition) tasks as much as possible⁷. Whereas, $\text{SPMC}(\delta = 4)$ outperforms WCR-EDF by as much as 12.2% for $U_B = 0.8$. As seen, the performance of SPMC increases as the δ increases but at the cost of increased preemption count.

Conventional MC scheduling algorithms drop all LC tasks upon mode switch. Hence, their schedulability performance would be significantly higher than those algorithms with LC support. Further, to evaluate such loss in schedulability performance we plot the acceptance ratio of the partitioned scheduling algorithm (EDF-VD [62] and AMC [71]). These algorithms use criticality-aware partitioning using UDP strategy [94], the best allocation strategy for these algorithms. We compare the loss in schedulability performance between the dynamic-priority algorithms (EDF-VD and SPMC) and likewise, between the fixed-priority algorithms (AMC and

⁷Although the performance improvement seems incremental, several WCR-EDF infeasible task sets are deemed schedulable by $\text{SPMC}(\delta = 1)$. Refer to Figure 6.5 for further details.

Semi-FP). As shown, the loss in schedulability between EDF-VD and SPMC is comparable to the loss in schedulability between AMC and Semi-FP.

6.6.3 Support for LC tasks on Dual-core Systems

Figure 6.3: LC Utilization Support for Dual-core Systems

We compare the percentage of additional LC utilization supported by SPMC over Semi-FP for $m = 2$ in Figure 6.3. We classify the schedulable task sets into three categories - schedulable only by SPMC($\delta = 1$), schedulable only by Semi-FP and schedulable by both algorithms. Since both SPMC and Semi-FP offer full service to LC tasks after one mode switch, we evaluate the supported LC utilization after two mode switch (*i.e.*, after both cores switch to the HC mode). As seen, SPMC($\delta = 1$) offers as much as 29.2% additional LC utilization compared to Semi-FP. On the contrary, Semi-FP offers only as much as 17.6% additional LC utilization. This is due to the algorithm property of dropping all the LC tasks on the core that switched last to the HC mode. For those task sets that are schedulable by both these algorithms, SPMC($\delta = 1$) offers 23.7% additional LC utilization compared to Semi-FP. Thus, SPMC offers better support for LC tasks in terms of total LC utilization after the mode switch.

6.6.4 Performance Evaluation for Varying Task Parameters

(a) $m = 2$ (b) $m = 4, 8$

Figure 6.4: Varying percentage of LC tasks

To evaluate the performance of the SPMC algorithm with respect to the percentage of LC tasks in a task set, we plot the weighted acceptance ratio (*i.e.*, acceptance ratio weighted with U_B) of the algorithm for varying $P_L \in \{0.1, \dots, 0.9\}$ and $m = \{2, 4, 8\}$ in Figure 6.4. Since Semi-FP is applicable only for $m = 2$, we first compare the performance of SPMC algorithm for varying percentage of LC tasks in a task set for $m = 2$ in Figure 6.4(a). Later, in Figure 6.4(b), we evaluate the performance of SPMC for $m = \{4, 8\}$.

As shown in Figure 6.4(a), the performance gap of the SPMC algorithm is higher when the percentage of LC tasks in a task set is lower. This is expected because when the task set comprises fewer LC tasks, it is easier to determine a destination core for migration. As shown in Figure 6.4(b), the performance of SPMC decreases initially and then increases with increasing number of LC tasks in a task set. The performance is high at the extremes as the task set comprises majorly of the same criticality tasks. Due to the algorithm property of partitioning all the HC tasks first the performance is low when the task set comprises both HC and LC tasks as these task sets are difficult to schedule using reservation for LC tasks.

6.6.5 Evaluation of Semi-Partitioned Scheduling

Figure 6.5: Percentage of WCR-EDF infeasible task sets that are schedulable by SPMC

To provide some insight into the effectiveness of semi-partitioned MC scheduling, we com-

pare the performance of SPMC with respect to partitioned scheduling algorithm WCR-EDF. We present the percentage of WCR-EDF infeasible task sets schedulable by the SPMC algorithm in Figure 6.5. As shown, for $m = 4$ and $U_B = 0.5$, as much as 16.3% more task sets are deemed schedulable by SPMC($\delta = 1$) and as much as 72.6% more task sets are deemed schedulable by SPMC($\delta = 4$). As the U_B value increases the performance of the SPMC algorithm decreases.

6.6.6 LC support for Varying Number of Mode Switch

Figure 6.6: Varying number of the mode switch

In Figure 6.6, we present the percentage of total LC utilization supported by the SPMC algorithm for varying number of the mode switch and $m = 8$. We plot the minimum guaranteed LC utilization for two different server periods ($\delta = 1$ and $\delta = 4$). As expected, the percentage of LC utilization supported decreases as the number of cores that switch to the HC mode increases. The LC utilization decreases initially and remains the same as the number of mode switch increases. The reason is that once all the cores with migrating LC tasks switch to the HC mode there is no further decrease in the LC support as the remaining cores are composed of only non-migrating LC tasks. This shows that even after more than one core switches to the HC mode the LC tasks are still supported under SPMC algorithm.

Thus, from our experiments, we can conclude that SPMC algorithm, even for the largest

server period, has better schedulability performance and better LC utilization support compared to existing algorithms that support full service to the LC tasks after single mode switch.

6.7 Conclusions

In this chapter, we presented the semi-partitioned mixed-criticality (SPMC) model to support the execution of LC tasks when the system switches to the HC mode. Unlike most existing approaches, we considered the problem of guaranteeing full service to the LC tasks when at most one processing core switches mode. Under the SPMC model, all the tasks are statically partitioned in the LC mode and upon mode switch, only the LC tasks are allowed to migrate between cores.

The SPMC model uses the reservation based technique to support the migration of LC tasks from one core to another. Upon mode switch, the migrating LC tasks in the source core (the core that switches to HC mode) are migrated to the server reserve in the destination core (the core to which these tasks migrate). As a result, as long as the destination core remains in the LC mode, the migrating LC tasks are executed with their LC execution budget. If the destination core also switches mode, then the LC tasks that migrated from another core are dropped and the migrating LC tasks native to the destination core are migrated to another core.

We presented the execution semantics of the SPMC model and derived the schedulability analysis. We also derived the system utilization bound which is used at design time for assigning tasks to cores. Then, we presented the bound on the number of preemption for utilizing the servers. Later, we proposed a partitioning strategy for the SPMC model. Finally, through experimental evaluation, we compared the performance of SPMC algorithm with the existing fixed-priority variant and showed SPMC significantly outperforms in terms of both schedulability and LC support.

Chapter 7

Conclusions and Future Work

In this chapter, we summarize the contributions of this thesis and present some possible future research directions. Section 7.1 presents the conclusions of our work. In Section 7.2 we identify the open problems and shortcomings in the state of the art research, and present some future research directions in multi-core MC scheduling.

7.1 Conclusions

Mixed-Criticality (MC) workload and scheduling model was developed to address the pessimism involved in the abstracted workload model such as WCET estimation which leads to under-utilization of system resources. The widely used MC workload model, for a given criticality level, allows only the tasks of same or higher criticality to execute. Based on this model, several techniques for scheduling MC systems on homogeneous multi-core platforms have been proposed. The initial studies considered the execution of different criticality tasks on different processing cores [75, 77, 99]. However, such execution is very pessimistic in terms of resource utilization. To overcome this, later studies considered the execution of multiple criticality tasks on the same processing core. Due to the complexity of the analysis, most of these studies only considered dual-criticality MC systems.

Based on the execution model of the tasks, multi-core MC scheduling can be classified into global and partitioned scheduling. Under global MC scheduling, tasks are allowed to execute on any of the processing cores and migrate between cores at runtime. In the case of partitioned MC scheduling, the tasks are statically assigned to a core and not allowed to migrate at runtime.

Some initial works on multi-core MC scheduling are based on global algorithms designed for classical real-time systems [27, 33]. However, these algorithms suffer from poor schedulability performance in the experimental evaluation and do not have a good theoretical performance bound for task systems that are hard to schedule. To overcome this problem, based on fluid scheduling, the dual-rate model was proposed for dual-criticality MC systems [28]. Experimental evaluation showed that the dual-rate model has the best schedulability performance among all existing global scheduling algorithms. This model has also been shown to have an optimal performance bound in terms of the speed-up factor [29]. Although the dual-rate model was shown to have the best schedulability performance and optimal speed-up factor, the performance of the dual-rate model deteriorates as the system utilization increases [78].

To address the above issue, in Chapter 4, we introduced the multi-rate fluid scheduling model. Under the multi-rate model, each HC task executes with multiple rates in the transition period after the mode switch. By varying the execution rates in the transition period we were able to accommodate a higher demand compared to the dual-rate model. Thus, improving the schedulability of the dual-rate model. We showed that the multi-rate model generalizes the dual-rate model if the multiple execution rates after the mode switch are all same, and has an optimal speed-up factor of $4/3$ similar to the dual-rate model. Experimental evaluation showed that our proposed model with multiple execution rates dominates the dual-rate model in terms of schedulability.

Although global MC scheduling algorithms improve the schedulability performance and have optimal performance bounds, their performance pales by comparison with partitioned MC algorithms. Based on this motivation, several partitioned algorithms have been designed for dual-criticality MC systems [32–35, 79, 80]. Partitioned MC algorithms are generally preferred in safety-critical industries due to their natural extension of single-core techniques. The partitioned MC scheduling comprises two problems - partitioning heuristic and single-core MC scheduling. Most existing partitioning heuristics for dual-criticality MC systems are very inefficient for the single-core MC scheduling algorithm applied on each core.

To overcome the abovementioned problem, in Chapter 5, we proposed a novel utilization difference based partitioning strategy for dual-criticality MC systems. Under this strategy, the additional demand of HC tasks after the mode switch is uniformly distributed among cores. By doing so, we were able to improve the performance of single-core MC scheduling algorithm. Experimental evaluation showed that performance improvement using our strategies is significantly higher compared to existing partitioned MC algorithms, and it increases as the number of

processing cores increases, depicting its scalability.

The MC workload model used in most studies make an unrealistic assumption that all the HC tasks simultaneously require additional execution budget upon mode switch and all the LC tasks can safely be dropped to sufficiently guarantee resources to the HC tasks. However, this is highly unlikely to occur in practice and very pessimistic in terms of quality of LC tasks service. To address this drawback, several techniques have been proposed for multi-core MC systems [56–58, 76, 82, 88, 89]. However, most of these studies either only guarantee full service for a subset of LC tasks or reduced service for all LC tasks after the mode switch. Since the probability that an HC task overruns its budget is very low in practice, it is very pessimistic to penalize some LC tasks completely or penalize all LC tasks uniformly. Ideally, it is better to execute these LC tasks with their LC execution budgets as long as possible.

Towards addressing the above issue, in Chapter 6, we introduced the semi-partitioned MC model for dual-criticality MC systems, wherein the LC tasks executing on a processing core are safely migrated upon mode switch to another processing core that has not experienced mode switch. By migrating these tasks to another core, we were able to support the execution of such LC tasks after mode switch for a longer possible duration. Our experiment results showed that the proposed technique was able to improve both the schedulability performance and total LC utilization supported than state-of-the-art MC scheduling algorithm.

Thus, this thesis has successfully addressed the important problems related to multi-core MC scheduling identified in Chapter 1.

7.2 Future Research Directions

In Section 7.1, we presented the contributions of this thesis related to multi-core MC scheduling. Nevertheless, several important research problems still remain open and we present some of them below.

Mixed-Criticality Feasibility Test. MC scheduling problem, even for conventional dual-criticality MC systems, has been known to be NP-Hard in the strong sense. Hence, the design of an optimal scheduling algorithm is very unlikely. Existing algorithms are only speed-up optimal and not optimal in general, *i.e.*, there exist several feasible task systems that are not schedulable by these algorithms. The platform capacity constraint gives the necessary feasibility condition for scheduling such systems (*i.e.*, $U_L^L + U_H^L \leq m$ and $U_H^H \leq m$ for homogeneous multi-core

platforms). And likewise, the sufficient feasibility condition is derived from the optimal speed-up factor (*i.e.*, $U_L^L + U_H^L \leq \frac{3m}{4}$ and $U_H^H \leq \frac{3m}{4}$). Since the exact feasible region of MC systems is not known, it becomes hard to quantify the performance of any MC scheduling algorithm for task systems that lie within these two bounds.

The multi-rate schedulability test presented in Chapter 4 tries to address this problem to some extent. It has been shown to have an optimal speed-up factor and at the same time have the best schedulability performance among existing algorithms. However, as shown in Chapter 4, Section 4.10.2, the multi-rate model is still a sub-optimal scheduling algorithm for dual-criticality MC systems. Hence, one important research direction for future is to determine the tight feasibility bound of dual-criticality MC systems.

Support for Execution of LC tasks. Support for the execution of LC tasks after the mode switch is an important problem that has been widely addressed (see Section 3.3). Most studies limit their attention to the execution of a subset of LC tasks or execution of all LC tasks with reduced execution budgets after the mode switch. The LC execution budget for HC tasks is sufficient for most execution scenarios and very rarely these tasks execute beyond their LC execution budgets. Hence, it is very pessimistic to penalize some LC tasks completely or all LC tasks uniformly. Moreover, not all HC tasks simultaneously demand additional execution budgets when any of the HC tasks executes beyond its LC execution budget. Ideally, it is better to execute all LC tasks with their actual execution budget as long as possible with minimal deadline misses after the mode switch, and increase the execution budget to only a subset of HC tasks that require them.

The semi-partitioned MC model proposed in Chapter 6 partly address this issue where all LC tasks are guaranteed schedulability as long as the total number of processing cores that switch mode is bounded by one. However, to sufficiently meet the deadlines of all migrating LC tasks enough core utilization is statically reserved and thus leading to under-utilization of processing resource. Since these are LC tasks, occasional deadline miss by these tasks do not impact the safety of the system. Therefore, one important future research direction is to remove the restriction of guaranteed full service to the LC tasks until one core switches mode and explore different migration models for LC tasks using reservations to improve the resource utilization. The propose semi-partitioned MC model also considers a system-level mode switch where all HC tasks simultaneously demand additional processing resource. Since this is highly unlikely to occur in practice, another possible research direction is to consider task-level mode switch rather than system-level mode switch. In particular, techniques such as MC-Adapt [44] have been developed

that consider task-level mode switch where only a subset of HC tasks switch to the HC mode. By considering task-level mode switch, rather than immediately migrating all LC tasks we can migrate them in batches. This will help to reduce the migration cost at runtime.

References

- [1] I. Lee and O. Sokolsky, “Medical Cyber Physical Systems,” in *CPS Demystified Session, Design Automation Conference (DAC)*, 2010.
- [2] J. W. S. W. Liu, *Real-Time Systems*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000.
- [3] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, 2nd ed. Springer Publishing Company, Incorporated, 2011.
- [4] “Military Standard 882C: System Safety Program Requirements,” January 1993.
- [5] A. K. Mok and D. Chen, “A multiframe model for real-time tasks,” *IEEE transactions on Software Engineering*, vol. 23, no. 10, pp. 635–645, 1997.
- [6] A. K.-L. Mok, “Fundamental design problems of distributed systems for the hard-real-time environment,” Ph.D. dissertation, Massachusetts Institute of Technology, 1983.
- [7] C. L. Liu and J. W. Layland, “Scheduling algorithms for multiprogramming in a hard-real-time environment,” *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973. [Online]. Available: <http://doi.acm.org/10.1145/321738.321743>
- [8] S. K. Baruah, “Feasibility analysis of recurring branching tasks,” in *Proceedings of Euromicro Workshop on Real-Time Systems*, 1998, pp. 138–145.
- [9] S. K. Baruah, “A general model for recurring real-time tasks,” in *Proceedings of IEEE Real-Time Systems Symposium*, 1998, pp. 114–122.
- [10] R. I. Davis and A. Burns, “A survey of hard real-time scheduling for multiprocessor systems,” *ACM Comput. Surv.*, vol. 43, no. 4, pp. 35:1–35:44, Oct. 2011. [Online]. Available: <http://doi.acm.org/10.1145/1978802.1978814>
- [11] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, “Proportionate progress: A notion of fairness in resource allocation,” in *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, ser. STOC ’93. New York, NY, USA: ACM, 1993, pp. 345–354. [Online]. Available: <http://doi.acm.org/10.1145/167088.167194>

- [12] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt, “DP-FAIR: A Simple Model for Understanding Optimal Multiprocessor Scheduling,” in *Real-Time Systems (ECRTS), 22nd Euromicro Conference on*, July 2010.
- [13] G. Nelissen, H. Su, Y. Guo, D. Zhu, V. Nélis, and J. Goossens, “An optimal boundary fair scheduling,” *Real-Time Systems*, vol. 50, no. 4, 2014.
- [14] S. Funk, G. Levin, C. Sadowski, I. Pye, and S. Brandt, “DP-Fair: a unifying theory for optimal hard real-time multiprocessor scheduling,” *Real-Time Systems*, vol. 47, no. 5, 2011.
- [15] P. Holman and J. H. Anderson, “Adapting pfair scheduling for symmetric multiprocessors,” *J. Embedded Comput.*, vol. 1, no. 4, pp. 543–564, Dec. 2005.
- [16] P. J. Prisaznuk, “Integrated modular avionics,” in *Aerospace and Electronics Conference (NAECON)*. IEEE, 1992, pp. 39–45.
- [17] “AUTOSAR Consortium. Automotive Open System Architecture (AUTOSAR),” <http://www.autosar.org>.
- [18] A. Burns and R. Davis, “Mixed criticality systems – a review,” <http://www-users.cs.york.ac.uk/burns/review.pdf>, 2018.
- [19] Radio Technical Commission for Aeronautics (RTCA), “Software Considerations in Airborne Systems and Equipment Certification,” http://www.rtca.org/store_product.asp?prodid=803.
- [20] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, “The Worst-Case Execution-Time Problem - Overview of Methods and Survey of Tools,” *ACM Transactions on Embedded Computing Systems*, vol. 7, 2008.
- [21] R. Wilhelm and J. Reineke, “Embedded Systems: Many Cores; Many Problems,” in *IEEE Symposium on Industrial Embedded Systems (SIES)*, 2012, pp. 176–180.
- [22] S. Vestal, “Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance,” in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, 2007, pp. 239–243.
- [23] S. Baruah, “Mixed criticality schedulability analysis is highly intractable,” 2009.
- [24] K. Agrawal and S. Baruah, “Intractability Issues in Mixed-Criticality Scheduling,” in *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), S. Altmeyer, Ed., vol. 106. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, pp. 11:1–11:21. [Online]. Available: <http://drops.dagstuhl.de/opus/volltexte/2018/8992>

- [25] B. Andersson, S. Baruah, and J. Jonsson, "Static-priority scheduling on multiprocessors," in *Real-Time Systems Symposium, 2001. (RTSS 2001). Proceedings. 22nd IEEE*, Dec 2001, pp. 193–202.
- [26] H. Li and S. Baruah, "Global Mixed-Criticality Scheduling on Multiprocessors," in *24th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2012.
- [27] R. Pathan, "Schedulability Analysis of Mixed-Criticality Systems on Multiprocessors," in *24th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2012.
- [28] J. Lee, K.-M. Phan, X. Gu, J. Lee, A. Easwaran, I. Shin, and I. Lee, "MC-Fluid: Fluid Model-Based Mixed-Criticality Scheduling on Multiprocessors," in *Real-Time Systems Symposium (RTSS), 35th IEEE International*, Dec 2014.
- [29] S. Baruah, A. Easwaran, and Z. Guo, "MC-Fluid: simplified and optimally quantified," in *36th IEEE International Real-Time Systems Symposium (RTSS)*, Dec 2015.
- [30] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.
- [31] O. R. Kelly, H. Aydin, and B. Zhao, "On partitioned scheduling of fixed-priority mixed-criticality task sets," in *2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, Nov 2011, pp. 1051–1059.
- [32] K. Lakshmanan, D. d. Niz, R. Rajkumar, and G. Moreno, "Resource allocation in distributed mixed-criticality cyber-physical systems," in *2010 IEEE 30th International Conference on Distributed Computing Systems*, June 2010, pp. 169–178.
- [33] S. Baruah, B. Chattopadhyay, H. Li, and I. Shin, "Mixed-criticality scheduling on multiprocessors," *Real-Time Systems*, vol. 50, no. 1, pp. 142–177, 2014.
- [34] P. Rodriguez, L. George, Y. Abdeddaim, and J. Goossens, "Multi-Criteria Evaluation of Partitioned Edf-Vd for Mixed-Criticality Systems Upon Identical Processors," in *Workshop on Mixed Criticality Systems (WMC)*, December 2013.
- [35] C. Gu, N. Guan, Q. Deng, and W. Yi, "Partitioned mixed-criticality scheduling on multiprocessor platforms," in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, March 2014.
- [36] S. Baruah, A. Burns, and Z. Guo, "Scheduling mixed-criticality systems to guarantee some service under all non-erroneous behaviors," in *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2016, pp. 131–138.
- [37] A. Burns and S. Baruah, "Towards a more practical model for mixed criticality systems," in *Workshop on Mixed Criticality Systems (WMC)*, Dec 2013, pp. 1–6.

- [38] D. d. Niz, K. Lakshmanan, and R. Rajkumar, "On the scheduling of mixed-criticality real-time task sets," in *30th IEEE International Real-Time Systems Symposium, 2009, (RTSS 2009. 30th IEEE*, Dec 2009, pp. 291–300.
- [39] T. Fleming and A. Burns, "Incorporating the notion of importance into mixed criticality systems," in *Proceedings of Workshop on Mixed Criticality Systems*, 2014, p. 33.
- [40] O. Gettings, S. Quinton, and R. I. Davis, "Mixed criticality systems with weakly-hard constraints," in *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, ser. RTNS '15. ACM, 2015, pp. 237–246. [Online]. Available: <http://doi.acm.org/10.1145/2834848.2834850>
- [41] P. Huang, P. Kumar, N. Stoimenov, and L. Thiele, "Interference constraint graph - a new specification for mixed-criticality systems," in *2013 IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA)*, 2013, pp. 1–8.
- [42] P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele, "Service adaptations for mixed-criticality systems," in *In Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2014.
- [43] M. Jan, L. Zaourar, and M. Pitel, "Maximizing the execution rate of low-criticality tasks in mixed criticality system," in *Proceedings of Workshop on Mixed-Criticality Systems*, 2013.
- [44] J. Lee, H. S. Chwa, L. T. X. Phan, I. Shin, and I. Lee, "Mc-adapt: Adaptive task dropping in mixed-criticality scheduling," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5s, pp. 163:1–163:21, Sep. 2017. [Online]. Available: <http://doi.acm.org/10.1145/3126498>
- [45] D. Liu, J. Spasic, N. Guan, G. Chen, S. Liu, T. Stefanov, and W. Yi, "Edf-vd scheduling of mixed-criticality systems with degraded quality guarantees," in *Proceedings of the IEEE Real-Time Systems Symposium*, 2016, pp. 35–46.
- [46] D. Liu, N. Guan, J. Spasic, G. Chen, S. Liu, T. Stefanov, and W. Yi, "Scheduling analysis of imprecise mixed-criticality real-time tasks," *IEEE Transactions on Computers*, 2018.
- [47] M. Neukirchner, P. Axer, T. Michaels, and R. Ernst, "Monitoring of workload arrival functions for mixed-criticality systems," in *Proceedings of the IEEE Real-Time Systems Symposium*. IEEE, 2013, pp. 88–96.
- [48] A. V. Papadopoulos, E. Bini, S. Baruah, and A. Burns, "Adaptmc: A control-theoretic approach for achieving resilience in mixed-criticality systems," in *30th Euromicro Conference on Real-Time Systems, ECRTS 2018*, 2018, pp. 14:1–14:22. [Online]. Available: <https://doi.org/10.4230/LIPIcs.ECRTS.2018.14>

-
- [49] F. Santy, L. George, P. Thierry, and J. Goossens, "Relaxing mixed-criticality scheduling strictness for task sets scheduled with fp," in *2012 24th Euromicro Conference on Real-Time Systems*, July 2012, pp. 155–165.
 - [50] H. Su and D. Zhu, "An elastic mixed-criticality task model and its scheduling algorithm," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '13. San Jose, CA, USA: EDA Consortium, 2013, pp. 147–152. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2485288.2485325>
 - [51] H. Su, N. Guan, and D. Zhu, "Service guarantee exploration for mixed-criticality systems," in *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Aug 2014, pp. 1–10.
 - [52] H. Su, D. Zhu, and S. Brandt, "An elastic mixed-criticality task model and early-release edf scheduling algorithms," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 22, no. 2, pp. 28:1–28:25, Dec. 2016. [Online]. Available: <http://doi.acm.org/10.1145/2984633>
 - [53] X. Gu, A. Easwaran, K.-M. Phan, and I. Shin, "Resource efficient isolation mechanisms in mixed-criticality scheduling," in *Proceedings of the Euromicro Conference on Real-Time Systems*, July 2015, pp. 13–24.
 - [54] X. Gu and A. Easwaran, "Dynamic budget management with service guarantees for mixed-criticality systems," in *Proceedings of the IEEE Real-Time Systems Symposium*. IEEE, 2016, pp. 47–56.
 - [55] D. de Niz and L. T. X. Phan, "Partitioned scheduling of multi-modal mixed-criticality real-time systems on multiprocessor platforms," in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2014, pp. 111–122.
 - [56] R. M. Pathan, "Improving the Quality-of-Service for Scheduling Mixed-Criticality Systems on Multiprocessors," in *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), M. Bertogna, Ed., vol. 76. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, pp. 19:1–19:22.
 - [57] J. Ren and L. T. X. Phan, "Mixed-criticality scheduling on multiprocessors using task grouping," in *27th Euromicro Conference on Real-Time Systems (ECRTS)*, July 2015, pp. 25–34.
 - [58] H. Su, D. Zhu, and D. Moss, "Scheduling algorithms for elastic mixed-criticality tasks in multicore systems," in *IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications*, Aug 2013, pp. 352–357.

- [59] W. A. Horn, "Some simple scheduling algorithms," *Naval Research Logistics Quarterly*, vol. 21, no. 1, pp. 177–185, 1974. [Online]. Available: <http://dx.doi.org/10.1002/nav.3800210113>
- [60] B. Kalyanasundaram and K. Pruhs, "Speed is as powerful as clairvoyance [scheduling problems]," in *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, Oct 1995.
- [61] H. Cho, B. Ravindran, and E. D. Jensen, "An optimal real-time scheduling algorithm for multiprocessors," in *27th IEEE International Real-Time Systems Symposium (RTSS'06)*, Dec 2006, pp. 101–110.
- [62] 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 *2012 24th Euromicro Conference on Real-Time Systems*, 2012, pp. 145–154.
- [63] A. Easwaran, "Demand-based scheduling of mixed-criticality sporadic tasks on one processor," in *2013 IEEE 34th Real-Time Systems Symposium*, Dec 2013, pp. 78–87.
- [64] P. Ekberg and W. Yi, "Bounding and Shaping the Demand of Mixed-Criticality Sporadic Tasks," in *24th Euromicro Conference on Real-Time Systems (ECRTS), 2012*, July 2012, pp. 135–144.
- [65] P. Ekberg and W. Yi, "Bounding and shaping the demand of generalized mixed-criticality sporadic task systems," *Real-Time Systems*, vol. 50, no. 1, pp. 48–86, Jan 2014.
- [66] J. Lee, S. Ramanathan, K. M. Phan, A. Easwaran, I. Shin, and I. Lee, "Mc-fluid: Multi-core fluid-based mixed-criticality scheduling," *IEEE Transactions on Computers*, vol. 67, no. 4, pp. 469–483, April 2018.
- [67] S. Baruah, H. Li, and L. Stougie, "Towards the design of certifiable mixed-criticality systems," in *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, April 2010, pp. 13–22.
- [68] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie, "Scheduling Real-Time Mixed-Criticality Jobs," *Computers, IEEE Transactions on*, vol. 61, no. 8, pp. 1140–1152, Aug 2012.
- [69] N. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times," 1991.
- [70] S. Baruah and A. Burns, "Implementing Mixed Criticality Systems in Ada," in *Proceedings of the Ada-Europe International Conference on Reliable Software Technologies*, 2011, pp. 174–188.

-
- [71] S. Baruah, A. Burns, and R. Davis, "Response-Time Analysis for Mixed Criticality Systems," in *Real-Time Systems Symposium (RTSS)*, 2011 IEEE 32nd, Nov 2011, pp. 34–43.
 - [72] A. Burns and R. I. Davis, "Adaptive mixed criticality scheduling with deferred preemption," in *Proceedings of the IEEE Real-Time Systems Symposium*, 2014, pp. 21–30.
 - [73] S. K. Baruah, V. Bonifaci, G. D'Angelo, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie, "Mixed-criticality Scheduling of Sporadic Task Systems," in *Proceedings of the 19th European Conference on Algorithms*, ser. ESA'11, 2011, pp. 555–566.
 - [74] A. Masrur, D. Müller, and M. Werner, "Bi-level deadline scaling for admission control in mixed-criticality systems," in *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2015, pp. 100–109.
 - [75] J. H. Anderson, S. K. Baruah, and B. B. Brandenburg, "Multicore Operating-System Support for Mixed Criticality," in *Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification*, 2009, Apr 2009.
 - [76] Z. Al-bayati, Q. Zhao, A. Youssef, H. Zeng, and Z. Gu, "Enhanced partitioned scheduling of mixed-criticality systems on multicore platforms," in *The 20th Asia and South Pacific Design Automation Conference*, Jan 2015, pp. 630–635.
 - [77] J. L. Herman, C. J. Kenna, M. S. Mollison, J. H. Anderson, and D. M. Johnson, "Rtos support for multicore mixed-criticality systems," in *2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*, April 2012, pp. 197–208.
 - [78] S. Ramanathan and A. Easwaran, "MC-Fluid: rate assignment strategies," in *Workshop on Mixed Criticality Systems*, 2015, Dec 2015.
 - [79] R. Gratia, T. Robert, and L. Pautet, "Scheduling of mixed-criticality systems with run," in *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, Sept 2015, pp. 1–8.
 - [80] J. J. Han, X. Tao, D. Zhu, and H. Aydin, "Criticality-aware partitioning for multicore mixed-criticality systems," in *2016 45th International Conference on Parallel Processing (ICPP)*, Aug 2016, pp. 227–235.
 - [81] A. Burns and S. Baruah, "Semi-partitioned cyclic executives for mixed criticality systems," in *Workshop on Mixed Criticality Systems (WMC)*, Dec 2015.
 - [82] M. A. Awan, K. Bletsas, P. F. Souto, and E. Tovar, "Semi-partitioned mixed-criticality scheduling," in *Architecture of Computing Systems - ARCS 2017*, J. Knoop, W. Karl, M. Schulz, K. Inoue, and T. Pionteck, Eds. Cham: Springer International Publishing, 2017, pp. 205–218.

- [83] H. Su, P. Deng, D. Zhu, and Q. Zhu, “Fixed-priority dual-rate mixed-criticality systems: Schedulability analysis and performance optimization,” in *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2016.
- [84] B. Hu, K. Huang, P. Huang, L. Thiele, and A. Knoll, “On-the-fly fast overrun budgeting for mixed-criticality systems,” in *Proceedings of the IEEE & ACM International Conference on Embedded Software*. IEEE, 2016, pp. 1–10.
- [85] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele, “Run and be safe: Mixed-criticality scheduling with temporary processor speedup,” in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2015.
- [86] F. Santy, G. Raravi, G. Nelissen, V. Nelis, P. Kumar, J. Goossens, and E. Tovar, “Two protocols to reduce the criticality level of multiprocessor mixed-criticality systems,” in *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, ser. RTNS ’13. New York, NY, USA: ACM, 2013, pp. 183–192. [Online]. Available: <http://doi.acm.org/10.1145/2516821.2516834>
- [87] I. Bate, A. Burns, and R. I. Davis, “A bailout protocol for mixed criticality systems,” in *2015 27th Euromicro Conference on Real-Time Systems*, July 2015, pp. 259–268.
- [88] H. Xu and A. Burns, “Semi-partitioned model for dual-core mixed criticality system,” in *Proceedings of the 23rd International Conference on Real Time and Networks Systems (RTNS)*. ACM, 2015, pp. 257–266.
- [89] H. Xu, *A Semi-partitioned Model for Scheduling Mixed Criticality Multi-core Systems*. PhD thesis, University of York, Nov 2017.
- [90] S. Ramanathan, A. Easwaran, and H. Cho, “Multi-rate fluid scheduling of mixed-criticality systems on multiprocessors,” *Real-Time Systems*, vol. 54, no. 2, pp. 247–277, Apr 2018. [Online]. Available: <https://doi.org/10.1007/s11241-017-9296-1>
- [91] R. McNaughton, “Scheduling with Deadlines and Loss Functions,” *Management Science*, vol. 6, no. 1, 1959.
- [92] S. Ramanathan and A. Easwaran, “Evaluation of Mixed-Criticality Scheduling Algorithms using a Fair Taskset Generator,” in *7th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, July 2016.
- [93] E. Paul, S. Roger, and D. Robert I., “Techniques for the synthesis of multiprocessor tasksets,” in *1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, July 2010.

- [94] S. Ramanathan and A. Easwaran, "Utilization difference based partitioned scheduling of mixed-criticality systems," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2017, pp. 238–243.
- [95] A. Burns and S. Baruah, *Timing Faults and Mixed Criticality Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 147–166.
- [96] S. Ramanathan and A. Easwaran, "Mixed-criticality scheduling on multiprocessors with service guarantees," in *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*, vol. 00, May 2018, pp. 17–24.
- [97] B. Andersson and K. Bletsas, "Sporadic multiprocessor scheduling with few preemptions," in *Euromicro Conference on Real-Time Systems (ECRTS)*, July 2008, pp. 243–252.
- [98] B. Andersson, K. Bletsas, and S. Baruah, "Scheduling arbitrary-deadline sporadic task systems on multiprocessors," in *Real-Time Systems Symposium (RTSS)*, Nov 2008, pp. 385–394.
- [99] M. S. Mollison, J. P. Erickson, J. H. Anderson, S. K. Baruah, and J. A. Scoredos, "Mixed-criticality real-time scheduling for multicore systems," in *Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology*, ser. CIT '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1864–1871.