1-1-2014

The Design, Analysis, & Application Of Multi-Modal Real-Time Embedded Systems

Masud Ahmed
Wayne State University,

THE DESIGN, ANALYSIS, & APPLICATION OF MULTI-MODAL REAL-TIME EMBEDDED SYSTEMS

by

MASUD AHMED

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2014

MAJOR: COMPUTER SCIENCE

Approved by:

Advisor	Date

DEDICATION

To my

MOTHER and FATHER

with love.

ACKNOWLEDGEMENTS

I am very grateful to my supervisor, Nathan W. Fisher of the Computer Science Department at Wayne State University, for his great advice, patient encouragement, consistent support, and also for insightful criticism. He has been the ideal thesis supervisor, and he has made the PhD possible. This dissertation had been supported by an NSF CAREER Grant (CNS-0953585), an NSF CSR Grant (CNS-1116787), and a grant from Wayne State University's Office of Vice President of Research.

I want to thank the members of my PhD dissertation committee, Dr. Weisong Shi and Dr. Hongwei Zhang of the Computer Science Department at Wayne State University, and Dr. Shengquan Wang of University of Michigan-Dearborn. Additionally, I am sincerely thankful to Wayne State University Computer Science Department professors and staff for their dedication, all the hard work, and for providing me the means to prepare myself for the future as a computer scientist.

I want to take this opportunity to thank my friend Mohammad Saiful Islam for inspiring me to pursue doctoral degree. I also want to thank Kazi Saidul Hasan, Mohammed Abul Monzur Azad, Mehrab Monjur, and Mohammad Ashrafuzzaman Khan for helping me through the graduate admission process. I am grateful to my CoPaRTS lab colleagues, Anwar Mohammadi, Guoyao Xu, Bo Peng, and Pradeep Hettiarachchi, for their continuous support. The opportunities to work with Dr. Moris Behnam in Summer 2009 and Dr. Joel Goossens in Summer 2012 were great honor for me. I owe to my friend Chowdhury Sayeed Hyder for helping me pass difficult phases of graduate life.

Finally, I would like to give my special grateful thanks to my mother (Masuda Sharif), my wife (Honey-Um-Maria), my sister (Fatema Islam), my brother (Tanvir Ahmed), my father-in-law (Abdur Rab Khan), and to all my family members for their love, support, encouragement, and for pushing me forward to make our dreams come true.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

CHAPTER 1: INTRODUCTION

1.1 Real-time System Overview

There are many systems (e.g., embedded systems, cyber-physical systems) which require timing constraints to ensure a certain quality-of-service. Research communities address these systems with timing constraints under a broad research area of *real-time systems*. In real-time systems, the correctness of a computation is dependent on the logical correctness of the results as well as the time at which these results are produced. Timing constraints of real-time systems are commonly specified in the form of deadlines within which individual activities should complete execution. For some systems, maintaining timing constraints are not as important as for some safety critical real-time systems (e.g., power plant). Based on the strictness of timing constraints, real-time systems may be categorized into two important groups: *hard real-time systems* and *soft real-time systems*.

For hard real-time systems, meeting deadlines of all activities are of supreme importance; failure to do so may cause critical failures and in some cases cause catastrophic hazard to human life ([2, 17]). A wide variety of applications use hard real-time systems including railway switching systems, automotive control systems (e.g., anti-lock braking systems and adaptive cruise control systems), flight control systems, space mission control, and nuclear power plants. When timing constraints are violated in power plant controllers, the plant could get overheated or even discharge radioactive substances into the surrounding environment. When deadlines are compromised in avionics, an airplane could lose control, potentially causing a catastrophic damage and losing invaluable human lives. Therefore, deadlines for these critical devices are indispensable in all circumstances.

In soft-real-time systems, the consequences of an occasional missed deadline are not always as severe as for hard real-time systems. Each soft real-time application may have specific timing requirements, and violating these timing requirements may result in undesirable consequences.

For instances, frequent deadline misses may degrade the system's appropriate quality of service. When timing requirements are violated in the case of cellular networks, ongoing calls could be dropped; if enough calls are dropped off frequently, the cell phone provider will lose valuable customers of their business. If an adaptive video streams frequently displays black screen in every event of system's property changes (e.g., change of network bandwidth over time), the user may lose patience for the internet service provider. As a result, these soft real-time systems may require certain performance guarantee (e.g., QoS) even in the presence of occasional deadline misses.

In addition to meeting all timing constraints for real-time systems, the ability to change system properties (e.g., processing capacity, network bandwidth, memory capacity) at runtime is a fundamental requirement of many real-time embedded devices for dynamic and efficient usage of device resources. For instance, an adaptive video-streaming application may change computation requirements due to changes in encoding/decoding requirements. A real-time control system may need to change system runtime environment in event of an unfavorable scenario. These changes in a system can be both application level and hardware capability. Different application levels sometimes are denoted as *software modes*, whereas different hardware capabilities of the same system are denote as *hardware modes*.

Modern processors support various power management features which can be effectively utilized for creating *hardware modes*. Dynamic voltage scaling (DVS) approach has become a standard capability for modern processors as a power management feature. DVS allows controlling the execution capability in the CPU from the operating system. Using DVS, user level application can change the processor's operating features, which in turn can reduce energy consumption as well as the production of heat. Intel and AMD processors (Intel SpeedStep® and AMDPowerNOW!^{TM}) support this technology. Some low-end processors support other techniques: auto-clock stop mode [7], enhanced auto-clock freeze mode [7], and auto-halt power-down mode [7]. These very basic features of low-end processors can play a vital role in managing devices processing resource efficiently.

Most of the real-time and embedded system devices use low-end processors due to manufacturing cost and volume constraints; therefore, these devices may not include the sophisticated dynamic-voltage-scaling capabilities of the higher-end processors. Low-power state [7], which is a basic power management feature, can be used conveniently for multiple hardware modes. Keeping the processor in a low-power state during idle periods as a DTM technique might lower instantaneous temperature [68] of the system, as well as the energy consumption. Meisner et al. [50] deployed similar techniques (with the name PowerNap) for eliminating idle powers in a large datacenter. They developed circuitry which would resume the processors from low-power mode if there is a activity detected by the NIC. However, erratically putting the processor in a low-power mode makes it extremely difficult to analyze the schedulability of hard real-time systems tasks.

Example of software modes can be found in server systems (e.g., databases, web server), intelligent personal assistant, and also in interactive design tools. Smartphones, digital video recorders, tablet computers, and many others support multiple simultaneously-executing adaptive multimedia streams on a shared computational platform. Different quality-of-service requirements for each multi-media stream may be modeled as software modes. For guaranteed QoS level for all participating streams, applications must ensure timing constraints during the execution of each mode. In addition, a transition period (i.e., duration between two modes) requires special attention as during the transition both the old-mode and the new-mode may be active; so, the processing resource may have to deal with transient workload which may be higher than the execution of a normal mode.

Additionally, many real-time systems may need to support multiple simultaneously-executing subsystems upon a shared computational platform. Upon such devices, guaranteeing a QoS level for co-executing applications is crucially important in many respects. In order to obtain a guaranteed service, applications upon a shared platform must be isolated from each other temporally which is frequently denoted as *temporal-isolation*. Temporal isolation is indispensable to ensure that each application receives an acceptable level of quality of service (QoS). However, in the presence of temporal isolation, simultaneously running applications essentially run on non-continuous processing resource. These non-continuous resources may be efficiently modeled by hardware modes [63].

In this thesis, we attempt to address multi-modal system schedulability considering both software/hardware modes that will provide a guarantee of service upon a shared computing platform. Before describing multi-modal systems in detail, the following subsections list some of the stakeholders which benefit directly from multi-modal system research.

1.2 Applications

1.2.1 Control Systems for Real-Time Computing

Control systems have the inherent ability of maintaining stability even in dynamic and unpredictable operating environments. Such control systems often require switching execution modes based on the feedback from observable variables (e.g., temperature, memory status, network status, and battery status) on the system. For developing a control system with hardware/software, system designers need to make sure that timing requirements for all real-time tasks are always fulfilled. Ensuring timing constraints during transition requires special attention as the system may get overloaded due to simultaneous presence of jobs from both old mode and new mode tasks. Therefore, determination of the minimum CPU requirements with guaranteed schedulability for multi-modal systems received special attention over the past few years. Figure 1.1 represents a control system where r is the desired output and y is the observed output from the plant. C, P, and F correspondingly denote controller, actuator/plant, and a set of sensors for measuring the output. In this dissertation, we address schedulability of a multi-modal system which are integral part of control-systems for real-time computing. The multi-modal model presented in this dissertation forms the basis of the design of a thermal-aware real-time system with predictable temporal and thermal performance.

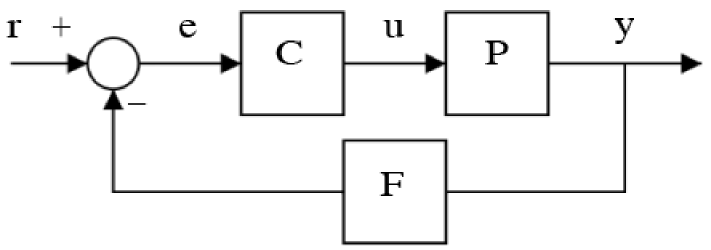

Figure 1.1: Components of a control computing system.

1.2.2 CPS

Cyber-physical systems are an integral part of industrial automation, automotive control systems, and power plants. As these systems exist in dynamic environments, the system must have the ability to change modes in order to adapt to different environmental contexts (e.g., an adaptive cruise control system may have different functional modes depending on travel conditions [61]). Ideally, each subsystem comprising a cyber-physical system could be allocated a dedicated continuously-available processing platform upon which *software mode changes* (i.e., changes in the underlying tasks, algorithms, or execution behavior) would provide the cyber support for controlling the physical plant in a dynamic environment. However, provisioning a dedicated and continuously-available processing platform for each subsystem may grossly over-provision the computational resources required to control the physical plant and increase the size, weight, and power of the system.

1.2.3 Satellite Systems

There are many types of embedded systems which could be found military and earth observation satellites, communications satellites [71], navigation satellites, weather satellites, and research satellites used for various purposes. To ensure the correct operation [19, 20], a satellite system may maintain many constraints including hard deadlines. To avoid unforeseen events, the satellite systems may deploy control systems for which possessing multiple operating modes is a fundamental requirement. Satellite communication may specially be benefitted from multi-modal systems as different types of communication (e.g., beacon, transponder, and repeater) may share hardware resource with varying frequency (uplink/downlink) requirements.

1.3 Multi-Modal System (MMS)

In this section, we present a short summary on contributions of this dissertation. First we discuss in brief different components of a multi-modal system (Subsection 1.3.1), research challenges (Subsection 1.3.2), thesis statement (Subsection 1.3.3), and finally how we address these challenges (Subsection 1.3.4).

1.3.1 Aspects of a Multi-Modal System (MMS)

Real-Time Workload

Workload is generated by the real-time application which typically associates a deadline for each activity. The firmness of the deadline depends on the underlying system. In this dissertation, to represent the recurring workload typical to control/CPS application, we model real-time workload by the *sporadic tasks model* [51]. A sporadic task $\tau_\ell^{(i)}$ is characterized by three parameters: *worst-case execution requirement* $e_\ell^{(i)}$, *(relative) deadline* $d_\ell^{(i)}$, and *minimum inter-arrival separation* $p_\ell^{(i)}$ (also called the *period*). The interpretation is that a sporadic task $\tau_\ell^{(i)}$ may produce a sequence of jobs, separated by at least $p_\ell^{(i)}$ time units, where each job has the maximum execution time $e_\ell^{(i)}$ units, and must complete within $d_\ell^{(i)}$ time units after its arrival (see Figure 1.2).

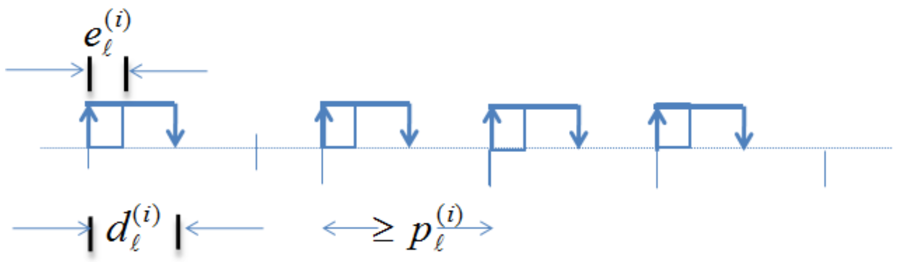

Figure 1.2: A possible job generation sequence of a sporadic task.

Processing Platform

In real time systems, a processing resource can be both uniprocessor and multiprocessor platform. In a uniprocessor resource, executions requirements are serviced through one execution unit (e.g., single core microprocessor, microcontroller). On the other hand, a multiprocessor platform may be either a multi-core processor or consist of more than one processor. In this dissertation, we target uniprocessor resource as the target platform for a real-time system; we plan to extend our techniques for uniprocessor resources to multiprocessor resources in future work. A processing platform may be completely occupied by a single application, or it may be shared by multiple applications at the same time. For the later case, a real-time scheduling algorithm may be required for sharing the processing resource among applications.

In order to reduce the manufacturing volume and cost, real-time system designers are more inclined to utilize low-end computing resources (e.g., microcontroller). As suggested earlier in the introduction, these devices may not have the sophisticated dynamic-power/thermal management (DPM/DTM) technology like DVS; however, very often these devices possess basic power management feature in the form of low-power state. Prior research work developed external circuitry (e.g., PowerNap [50]) to utilize these basic power management features to put the processing resource in a low-power state opportunistically. EDP resource model can address such discontinuous processing behavior with a higher degree of optimism.

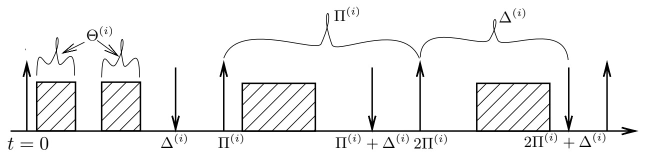

Figure 1.3: A possible execution pattern of an EDP resource.

In the EDP resource model, a processing resource $\Omega^{(i)}$ is represented by $\Pi^{(i)}$, $\Theta^{(i)}$, and $\Delta^{(i)}$ where $\Pi^{(i)}$ is the *resource period*, $\Theta^{(i)}$ is the *resource capacity*, and $\Delta^{(i)}$ is the *resource deadline*. The interpretation is that the processor will be executed in active-mode only Θ units of time in each successive $\Pi^{(i)}$-length intervals. For EDP, processing allocation must occur within $\Delta^{(i)}$ time the start of the resource period. Figure 1.3 shows the execution pattern. In this dissertation, we use the EDP model as the basis for representing a hardware mode.

Scheduling Algorithms

Scheduling algorithm determines which job gets the hardware at any instant of time. For ensuring all the timing constraints (in the form of deadlines of jobs), a deterministic scheduling algorithm is important. Most scheduling algorithms operate as follows: at any time instant, each active job is assigned a priority, and the scheduling algorithm chooses for execution the currently active job with the highest priority. Among the scheduling algorithms studied in real-time systems are divided into two important groups: Dynamic-Priority scheduling and Fixed-Priority scheduling. The example of dynamic priority includes Earliest Deadline First (EDF) [73], Least Laxity First [30], Pfair-based algorithms [13], and Earliest Deadline with Zero Laxity [27]. Rate Monotonic [49] and Deadline Monotonic [12] are the example of fixed-priority algorithms. In this dissertation, we consider EDF and FP for the multi-modal system.

Schedulability Analysis

Schedulability analysis is an essential tool for real-time system design. Schedulability analysis for a particular scheduling algorithm takes a real-time multi-modal system as an input, and checks whether all timing constraints are met under the particular algorithm. The analysis returns "Yes"/"No" response by analyzing the real-time workload and processing resource. If the response is "Yes", then the real-time system will not miss any deadline under any circumstance; however, if the response is "No", then the real-time system may miss a deadline during its execution. To decide schedulability of a multi-modal systems, the schedulability analysis must evaluate all modes, and during the transition between any pair of modes. Therefore, schedulability analysis is computationally expensive. In this dissertation, we consider FP and EDF schedulability analysis of multi-modal system for ensuring timing constraints at design time.

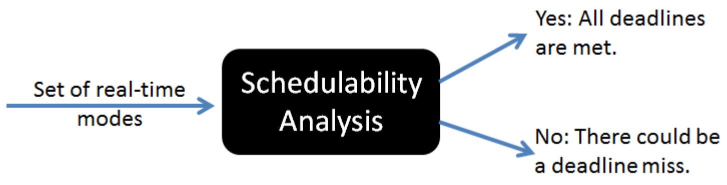
Set of real-time modes
Schedulability Analysis
Yes: All deadlines are met.
No: There could be a deadline miss.

Figure 1.4: Input/output of a schedulability analysis.

1.3.2 Research Challenges for Multi-Modal Real-Time Systems

While multiple modes enable flexibility and increase dynamism, switching modes may result in severe performance degradation (e.g., temporary black screen while changing power savings modes in laptops and smartphones). For systems that maintain performance guarantees using recurring tasks with deadlines, a smooth transition with timing constraints is possible using a schedulability analysis. Traditional research on real-time systems has commonly addressed the issues of schedulability under mode changes and temporal isolation, separately and independently. On the other hand, research on temporal isolation in real-time scheduling (often called *server-based* or *hierarchical* scheduling), while permitting the analysis of real-time subsystems that co-execute upon a shared computation platform, has often assumed that the application and resource requirements of each subsystem are fixed during runtime. Recently researchers have started to address the problem of guaranteeing hard deadlines of temporally-isolated subsystems in multi-modal systems. However, most of this recent research suffers from two fundamental drawbacks:

1. Full support for both resource-level mode changes and application-level mode-changes does not exist.

2. The proposed algorithms for determining schedulability under mode changes have exponential-time complexity.

Full support for both resource-level and application-level mode-changes are important to achieve temporal isolation while executing multiple **hard** real-time systems (i.e., subsystems) upon a shared platform. Individual support for either resource-level or application-level mode-changes is not effective enough for their desired purpose with the absence of another. For example, application-level only mode-changes cannot exploit modes with lighter resources without presence of hardware modes. On the other hand, hardware resource mode is not usable if the application does not possess the capability of utilizing them. Schedulability analysis developed independently may not provide real-time guarantees where both hardware-level and application-level modes are present. So in this thesis, we develop schedulability analysis for multi-mode system considering both hardware and software modes together.

1.3.3 Thesis

The thesis of this dissertation is:

> *The determination of resource parameters with guaranteed schedulability for real-time systems that may change execution requirements over time is computationally expensive. However, decoupling schedulability analysis from determining the minimum processing-resource parameters of a real-time multi-modal system results in the pseudo-polynomial complexity for the combined goals of determining both MMS schedulability and optimal resource parameters.*

1.3.4 Contribution and Organization

To support this thesis, this dissertation makes the following contributions:

- Using the theoretical analysis for characterizing the peak-temperature of a periodic resource, we show the benefit of intelligently choosing parameters of a multi-modal system in Chapter 4. This analysis suggests that a periodic resource with higher ratio of capacity to period-of-repetition generates a higher peak-temperature.

- We propose the *discrete hardware/software real-time multi-mode* model in Chapter 5 suitable for real-time embedded system that may need to maintain certain quality-of-service in changing environments. The protocol is also suitable for real-time control system which would ensure predictable system behavior in both favorable and unfavorable scenarios.

- We derive a sufficient schedulability analysis test for the setting where the sequence of mode changes is a priori fixed and each mode has its workload scheduled by the earliest-deadline-first (EDF) scheduling algorithm [49] in Chapter 6. This setting is referred to as *concrete* mode-change request setting.

- We derive a pseudo-polynomial schedulability analysis test for the setting where mode changes are not a priori fixed (called *non-concrete* sequences) and each mode is scheduled by EDF in Section 6.2. The time complexity of our proposed test is a significant improvement over previous tests that require exponential-time in the worst case. In Chapter 7, we extended the schedulability analysis for fixed-priority tasks.

- In Chapter 8, we develop MinMaxCap algorithm to minimize resource usages of a multi-modal system. We are not aware of any result that addresses the optimality in terms of resource usages with respect to any objective function. We exploit the schedulability analysis (Chapter 6) developed in this thesis to obtain set of optimized hardware resource modes ensuring system schedulability.

- To accelerate the schedulability analysis of a multi-modal system as well as determination of minimum parameters, we developed a parallel algorithm suitable for message-passing parallel systems to check the invariants (Chapter 9) of the schedulable real-time MMS. To improve further, we develop algorithm for checking schedulability of MMS using massively parallel graphical processing unit (GPU) platform.

The thesis of this dissertation is motivated by the requirements of control computing under real-time setting. A computing control system is very convenient in achieving predictability and stability in unfavorable scenarios. A multi-modal system is a fundamental requirement of a computing control system. Each of the contributions of this thesis addresses an issue related to the development of a real-time multi-modal system for control computing upon a single processing resource. Multicore processors are very common and available for embedded computing; so an interesting topic for future research will be multi-modal systems upon multicore CPU.

CHAPTER 2: MODELS & DEFINITIONS

In this thesis, we are interested in analyzing characteristics of real-time multi-modal systems. For making the thesis more readable, proper definitions of used terms are very much essential. This chapter presents definitions and notations for the task model, workload functions, and periodic resource model that were used throughout the thesis. At the end of this chapter, we discuss different components of a multi-modal system.

2.1 Definitions

2.1.1 Jobs

In real-time systems, there are basic units of work known as jobs, which need to be executed by the processing unit. Each such job has a deadline. In this thesis, only preemptive model of scheduling is addressed; that is, a job executing on the processor may be interrupted, and its execution resumed at a later point in time. For the ease of presentation, we assume that there is no penalty for preemption. (Bertogna et al. [15] and Baruah [11] developed techniques to address preemption which can be integrated into the techniques of this dissertation.) Each real-time job is characterized by three parameters: a release time, an execution requirement, and a deadline; with the interpretation that the job needs to be executed for an amount equal to its execution requirement between its release time and its deadline.

Definition 1 (Job) *A real-time job $j = (a, e, d)$ is characterized by three parameters an arrival time a, an execution requirement e, and a deadline d, with the interpretation that this job must receive e units of execution over the interval $[a, d)$.*

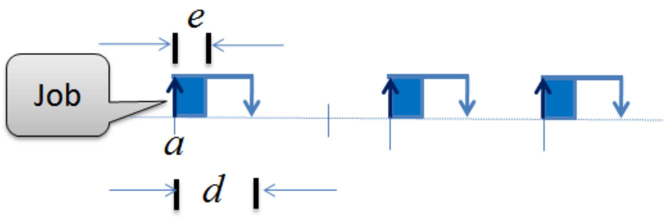

Figure 2.1: Real-time jobs with same execution requirement e.

2.1.2 Tasks

A real-time system is usually modeled as a set of concurrent *tasks*. Each task generates an infinite succession of jobs. There are different models available for defining a task. For this report, we consider the sporadic task model, and we define sporadic tasks systems in the next subsection.

2.1.3 Scheduling Algorithms

Most scheduling algorithms operate as follows: at any instant of time, an active job is assigned a priority, and the scheduling algorithm chooses for execution the currently active job with the highest priority. This priority assignment can statically assigned to each task at the design time (i.e., fixed-priority), or task priority can be changed dynamically. Among the scheduling algorithms studied in real-time systems are: Earliest Deadline First (EDF) [73], Rate Monotonic [49], Deadline Monotonic [12], Least Laxity First [30], Pfair-based algorithms [13], and Earliest Deadline with Zero Laxity [27]. In this thesis, we consider EDF and fixed-priority (FP) scheduling algorithm.

§**EDF.** The earliest-deadline-first (EDF) is a dynamic scheduling algorithm. At each instant of time, EDF chooses for execution the currently-active job with the smallest deadline. EDF is an optimal scheduling algorithm for scheduling arbitrary collections of independent real-time jobs in the following sense - if there exists a schedule for a given collection of independent preemptible jobs such that meet all their deadlines, then the schedule generated by EDF for this collection of jobs will meet their all deadlines too.

§**FP.** The fixed-priority scheduling of periodic and sporadic task systems, all the jobs generated by each task are required to be the same priority, which should be different from the priorities assigned to jobs generated by other tasks in the system. Hence, the run-time scheduling problem essentially reduces to the problem of associating a unique priority with each task in the system.

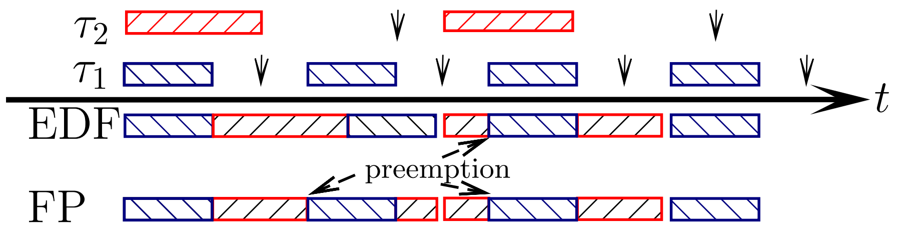

Figure 2.2: FP vs. EDF schedule for two tasks τ_1 and τ_2.

Figure 2.2 shows EDF and FP schedule for a same set of jobs. Forward diagonally hatched rectangles are representing execution requirements for jobs of τ_2 whereas backward diagonally hatched rectangles are representing jobs of τ_1. Downward arrows point deadlines for corresponding jobs. For FP, task τ_1 is assumed higher priority than τ_2. Job arrival sequences for both tasks are depicted above the timeline whereas job execution schedules for EDF and FP are shown below the timeline. As shown in the figure, due to priority of individual job, the EDF schedule has one preemption whereas FP schedule contains two preemptions.

Before we further describe the scheduling approaches, we define some terms commonly used in describing properties of real-time scheduling algorithms.

1. **Feasible**: A schedule is said to be feasible if all jobs from each tasks are completed according to timing constraints.

2. **Schedulable**: A set of tasks is said to be schedulable if a scheduling algorithm can produce a feasible schedule.

3. **Optimal**: A scheduling algorithm is said to be an optimal if the algorithm is able to produce a feasible schedule for any schedulable task set.

2.2 Models

2.2.1 Explicit-Deadline Periodic (EDP) Resource Model

The *EDP resource model* [32, 70] is a general resource model for characterizing the execution of a system upon a periodically-available and non-continuously-executing resource. The processing resource available to each mode $M^{(i)}$ is represented by an EDP resource $\Omega^{(i)} \equiv \left(\Pi^{(i)}, \Theta^{(i)}, \Delta^{(i)}\right)$ where $\Pi^{(i)}$ is the *period-of-repetition* (also known as resource-period), $\Theta^{(i)}$ is the *resource capacity*, and $\Delta^{(i)}$ is the *resource deadline*. The interpretation of these parameters is that the EDP resource $\Omega^{(i)}$ guarantees mode $M^{(i)}$ a total execution of at least $\Theta^{(i)}$ units over successive $\Pi^{(i)}$-length intervals within $\Delta^{(i)}(\leq \Pi^{(i)})$ units of time. Furthermore, we assume that EDF is used to schedule the workload at any point when the resource is providing execution. The ratio $\Theta^{(i)}/\Pi^{(i)}$ is frequently denoted as *interface-bandwidth* $I^{(i)}$. Real-time researchers utilize supply-bound function to quantify the minimum supply at any interval.

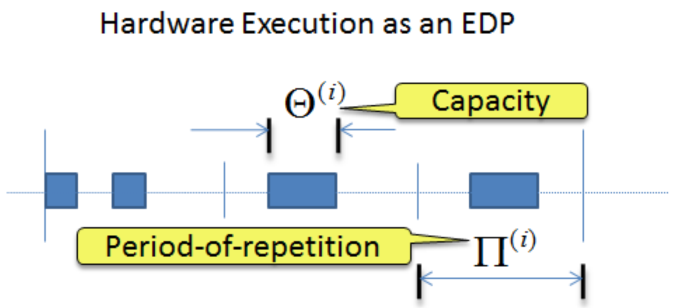

Figure 2.3: EDP resource with period-of-repetition equal to deadline.

Definition 2 (Supply-Bound Function) *For any $t > 0$, the* **supply-bound function** $\mathsf{sbf}(\Omega^{(i)}, t)$ *quantifies the minimum execution supply that a mode $M^{(i)}$ is guaranteed to receive from $\Omega^{(i)}$ over any interval of length $t \geq 0$. Easwaran et al. [32] have quantified $\mathsf{sbf}(\Omega^{(i)}, t)$ as follows:*

$$\mathsf{sbf}(\Omega^{(i)}, t) = \begin{cases} y\Theta^{(i)} + \max\left(0, t - x - y\Pi^{(i)}\right), & \text{if } t \geq \Delta^{(i)} - \Theta^{(i)} \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

where $y = \left\lfloor \frac{t-(\Delta^{(i)}-\Theta^{(i)})}{\Pi^{(i)}} \right\rfloor$ *and* $x = (\Pi^{(i)} + \Delta^{(i)} - 2\Theta^{(i)})$. *The solid line in Figure 2.4 shows supply-bound function of* $\Omega^{(i)}$ *for the corresponding interval length.*

2.2.2 Task Model

A *sporadic task system* $\tau^{(i)} \stackrel{\text{def}}{=} \{\tau_1^{(i)}, \ldots, \tau_{n_i}^{(i)}\}$ is a collection of n_i sporadic tasks where each task $\tau_\ell^{(i)} \in \tau^{(i)}$ is characterized by three parameters: *worst-case execution requirement* $e_\ell^{(i)}$, *(relative) deadline* $d_\ell^{(i)}$, and *minimum inter-arrival separation* $p_\ell^{(i)}$ (also called *period*). A sporadic task $\tau_\ell^{(i)}$ may produce a sequence of jobs, separated by at least $p_\ell^{(i)}$ time units, where each job has the maximum execution time $e_\ell^{(i)}$ units, and must complete within $d_\ell^{(i)}$ time units after its arrival. We consider *constrained deadline* tasks; that is, $d_\ell^{(i)} \le p_\ell^{(i)}$. The utilization of a task $\tau_\ell^{(i)}$ is defined as $u_\ell^{(i)} \stackrel{\text{def}}{=} e_\ell^{(i)}/p_\ell^{(i)}$ and the task system utilization $u^{(i)}$ equals $\sum_\ell u_\ell^{(i)}$.

2.2.3 Workload Functions

A sporadic task $\tau_\ell^{(i)}$ specify only the minimum inter-arrival separation between jobs using the parameter $p_\ell^{(i)}$; therefore, there can be infinite combinations of valid job sequence possible for a single sporadic tasks. Real-time system research usually looks for the worst-case job arrival sequence, and develops **demand-bound function** to quantify the worst case execution demand and **request-bound function** to quantify the worst case execution requests over any interval.

Definition 3 (Demand-Bound Function) *For any* $t > 0$ *and task* $\tau_\ell^{(i)}$*, the* **demand-bound function** $\mathsf{dbf}(\tau_\ell^{(i)}, t)$ *quantifies the maximum cumulative execution requirement of all jobs of* $\tau_\ell^{(i)}$ *that could have both the arrival time and the deadline in any interval of length* t.

Baruah et al. [10] have shown that the demand-bound function for sporadic tasks can be calculated in a very similar way for that of strictly periodic tasks as follows:

$$\mathsf{dbf}(\tau_\ell^{(i)}, t) = \max\left(0, \left\lfloor \frac{t - d_\ell^{(i)}}{p_\ell^{(i)}} \right\rfloor + 1\right) \cdot e_\ell^{(i)}. \tag{2.2}$$

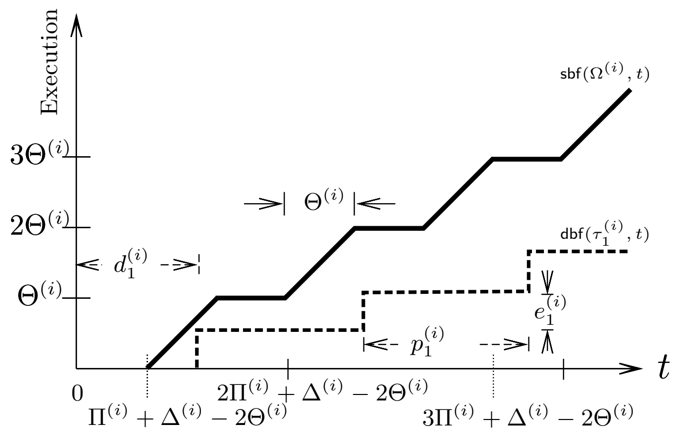

Figure 2.4: Visual depiction of the demand-bound function for a sporadic task and supply bound function for a periodic resource.

Figure 2.4 gives a visual depiction of the demand-bound function for a sporadic task $\tau_\ell^{(i)}$. The step function denotes a plot of $\mathsf{dbf}(\tau_\ell^{(i)}, t)$ as a function of t. The dashed line represents supply-bound function $\mathsf{sbf}(\Omega^{(i)}, t)$. As defined in the above definition and Figure 2.4, the dbf is a right continuous function with discontinuities at time points of the form $t \equiv d_\ell^{(i)} + a \cdot p_\ell^{(i)}$ where $a \in \mathbb{N}$. It has been shown [10] that the condition $\mathsf{dbf}(\tau_\ell^{(i)}, t) \leq \mathsf{sbf}(\Omega^{(i)}, \forall t \geq 0$ is necessary and sufficient for a sporadic task system τ to be EDF-schedulable (under preemptive settings) upon a uniprocessor platform of unit speed.

Definition 4 (Request-Bound Function) *For any $t > 0$ and task $\tau_\ell^{(i)}$, the* **request-bound function** $\mathsf{rbf}(\tau_\ell^{(i)}, t)$ *quantifies the maximum cumulative execution requirements of all jobs of $\tau_\ell^{(i)}$ that can have the arrival time in any interval of length t. Lehoczky et al. [47] have shown that for a sporadic task $\tau_\ell^{(i)}$, the* rbf *can be calculated as* $\mathsf{rbf}(\tau_\ell^{(i)}, t) = \left\lceil t/p_\ell^{(i)} \right\rceil \cdot e_\ell^{(i)}$.

2.3 A Multi-Modal System

In this section, we discuss different concepts of multi-modal systems. A system can have multiple modes (e.g., Initialization mode, Check mode, Emergency mode, Alarm mode, Fault recovery mode, etc.) for which the behavior of the system and processing resource may vary from each other. In order to ensure a constraint, a mode may specify both its software (real-time workload of tasks) and the corresponding hardware requirements (characterized by periodic resources). At any instant of time, the multi-mode system either executes in only one of its modes, or the system is in a transition between modes. The system may switch between different modes. We denote this event as a mode-change request (MCR).

2.3.1 Mode-Change Request

A mode-change request is an event that triggers a transition to a new mode from the currently executing mode. The mode that a system executed before the mode-change is defined as *old-mode* or *origin-mode*, and the newly arrived mode is known as *new-mode* or *destination-mode*.

A mode-change request can be either an external event or an interval event. Whenever a system detects a mode-changing event, the system initiates the process using a mode-change request. The system then switches to a new-mode. This switching between modes may result a transient state with additional workload. System designer occasionally provisions optional transition period. The length of transition period after a mode-change request is known as the *offset*. For a successful mode-change, we consider following three properties:

1. **Schedulability**: the multi-modal system must ensure all deadline constraints during the transition as well as during the normal execution of any mode.

2. **Periodicity**: the multi-modal system must provision some important tasks to execute without any effect from the mode-change request.

3. **Promptness**: for some important new mode tasks (e.g., those are in emergency mode), the system must finish execution within their deadlines.

The term "periodicity" and "promptness" are defined in Real and Crespo [64]. After mode change, system designers occasionally support delaying new-mode tasks to start a fixed duration. This transition period is known as *offset*.

2.3.2 Mode-Change Protocols

After a mode-change request, the system may have to change both software (i.e., tasks set) and hardware (i.e., periodic-resources) modes. Some less important tasks would be aborted, and some important new-mode tasks may need to start immediately. These changes are done according tom *mode-change protocols*. Based on the promptness of new-mode added tasks, Real and Crespo [64] divided the protocols for mode-changes into two groups as follows:

1. **Synchronous protocols**: In a synchronous mode-change protocol, the new mode-tasks cannot start execution as long as there are unfinished jobs from old-mode tasks.

2. **Asynchronous protocols**: In an asynchronous mode-change protocol, new mode tasks are allowed to execute with old-mode tasks.

The way the system may handle common tasks between old-mode and new-mode tasks, the protocols may be divided into two groups:

1. **Protocols with periodicity**: In a mode-change protocol with periodicity, common tasks between new-mode and old-mode execute independent of mode-change request.

2. **Protocols without periodicity**: In a mode-change protocol without periodicity, common tasks between new-mode and old-mode may be suspended at the time of mode-change request. These suspended tasks may resume its execution after the transition period.

2.3.3 Variation of Tasks

At the time of mode-change request, tasks may have different importance. Based on the importance, tasks may be classified into categories as follows:

1. **Aborted tasks**: There may be some less important tasks which may be removed from the system immediately at the time of mode-change request. We denote these tasks as aborted tasks.

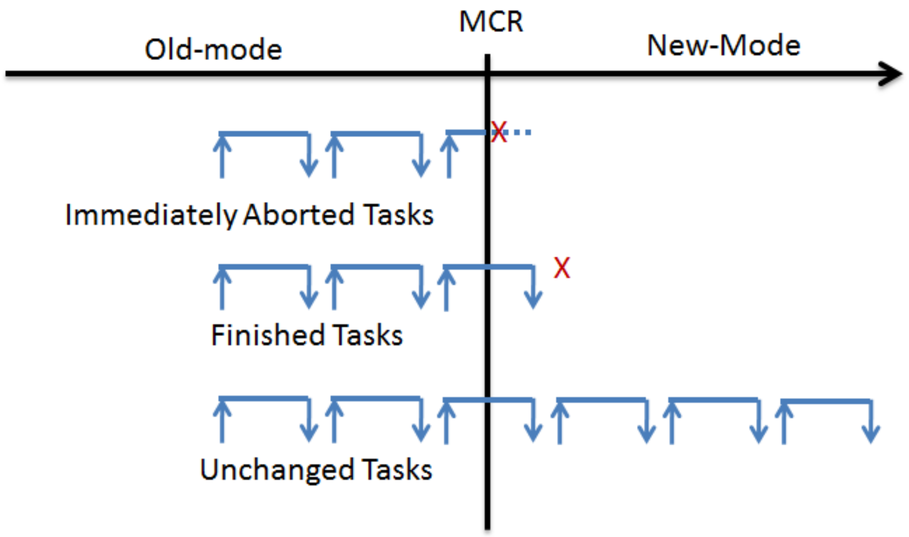

Figure 2.5: Tasks at the time a mode-change request.

2. **Finished tasks**: Immediate removal of some tasks at the time of mode-change request may leave the system at an inconsistent state. We denote these tasks as finished tasks for which the last job at the time of mode change request is allowed to finish its execution.

3. **Unchanged tasks**: Some important tasks that are common between old-mode and new-mode may need to continue without any effect from the mode-change request for the correct operation. We denote these tasks an unchanged tasks.

With the definition presented in this chapter, this thesis particularly emphasizes unchanged tasks and aborted tasks with mode-change protocols for real-time systems scheduled using either EDF or FP. We consider promptness of a mode-change request which can be achieved by allowing pre-calculated offset (e.g., transition period) after a mode-change request. Our proposed mode-change protocol will allow periodicity for unchanged tasks and finished tasks during/after a mode-change request. This objective is motivated by the requirements of predictable and stable computing in cyber-physical systems using control computing developed on top of real-time multi-modal systems.

CHAPTER 3: LITERATURE SURVEY

In this chapter, we present prior research used for real-time multi-modal systems. Some of these techniques are limited to work only with either software or hardware modes, others may exploit underlying hardware resources (e.g., software and hardware modes are coupled). We present previous research on these both kinds of systems along with processing resources, mode-change protocols, and serial schedulability analysis related to multi-modal systems. In the thesis, we present also expedited schedulability analysis using parallel computing, which is especially beneficial while optimizing resource usages of a multi-modal system; therefore, we include a summary of previous work on both parallel schedulability analysis and multi-modal system at the end of this chapter.

3.1 Processing Resources of MMS

Low-end microprocessors and micro-controllers, which are prevalent in today's industry automation ranges from toys to nuclear power plant controllers, help managing factories, guide weapon systems, and ensure flow of information worldwide. This is partially due to the fact that in many cases the design of a real-time system is constrained by the volume, size, and the packaging cost requirements of an embedded system. In real-time system research, these low-end processors as well as non-continuous usages of modern processors (due to sharing processing resources among applications) can be modeled using periodic-resources [32]. Periodic resources can inherently support discrete execution capabilities, which enable us to design hardware modes to be incorporated with software modes and mode-change protocols. There have been recent research on real-time systems targeted for controlled usages [42] of processing capability/power on top of periodic-resources for both achieving stability and temporal isolation while sharing processing resources among applications. The next section addresses mode-change protocols which define the behavior of a system at the time of a mode-change request.

3.2 Mode-Change Protocols

Numerous mode-change protocols exist for ensuring timing guarantees during transitions between modes on both uniprocessor [60, 64, 67, 77] and multiprocessor [54, 55] systems under the assumption of a dedicated processing platform. For this thesis, we restrict our attention to the uniprocessor setting. For application-level mode-changes, Tindell et al. [76] introduced a simple protocol where new-mode tasks wait until the processor finishes jobs from old-mode tasks. This approach is known as a synchronous mode-change protocol. Tindell et al. [77] defined a closed-form expression for calculating the waiting time (also known as offset) after which a new-mode task can generate jobs. Pedro et al. [60] and Real et al. [64] explored asynchronous mode-change protocols (i.e., old-mode tasks may execute concurrently with new-mode tasks) and determined the effects of introducing an offset during a mode-change on the schedulability. Guangming [40] studied the problem of calculating the best time to introduce new tasks into an EDF-scheduled system.

For ensuring temporal isolation between real-time subsystems co-executing on the same processing platform, numerous server-based frameworks have been proposed (e.g., constant-bandwidth server (CBS) [1], sporadic server [72], periodic-resource model [70], and bounded-delay resource-partitions [34]). However, most of these frameworks and their associated schedulability analysis assume that the application and resource requirements of the subsystems executing upon the server are priori fixed before the actual deployment. Subsequent work has attempted to remove this assumption. Frameworks such as elastic scheduling [21] and rate-based earliest deadline (RBED) [18] permit a subsystem to change its application or resource requirements adaptively; however, each of these previous results are soft real-time; that is - they do not guarantee all deadlines are met under transitions between executing modes. An adaptive hard real-time extension of CBS, called variable-bandwidth server (VBS) has been developed; however, VBS does not consider resource-level mode-changes and does not permit arbitrary mode-changes. (Instead, an application mode-change must pass an admission control test and be deferred until the mode-change is safe).

In the past two years, there has been increased research attention on developing hard real-time frameworks and analysis that support temporal isolation, and allow both resource and application mode-changes. Stoimenov et al. [74] developed a real-time calculus (RTC) approach for analyzing the application demand of a subsystem during a single application-level mode-change. In a complementary paper, Santinelli et al. [66] developed an RTC characterization of the resource supply of a resource-level mode-change. Taken together, these two results can be used to analyze the schedulability of a subsystem *under a single mode-change*; however, the results, in general, do not address carry-in; therefore, the results do not hold for a subsystem that might go through successive mode-changes. The result may be used with multiple successive mode-changes, only if there exists a subsystem idle time between each mode-change; the drawback is that such an approach is very conservative as the subsystem may have to delay a mode-change for a long duration until an idle time is found. In a later paper, Stoimenov et al. [75] investigated resource-level mode-changes in a Time Division Multiple Access (TDMA) server; however, schedulability analysis for application-level mode-changes was not investigated. Inam et al. [44] implemented simple mode-change protocol upon FreeRTOS for systems scheduled hierarchically. For this work, a task executing through mode-change requests without interruption (i.e., unchanged task) is not possible. None of the schedulability analysis in the aforementioned description have known tractable time complexity. (The authors do not mention the time complexity.) In contrast, our approach addresses successive application/resource mode-changes and has pseudo-polynomial time complexity.

One set of recent result has addressed multiple resource and application-level mode-changes (without waiting for idle times); Phan et al. [62, 63] proposed a general compositional model and associated analysis techniques for processing multiple bursty/complex event/data streams using state-based models such as timed automaton. However, as the analysis requires traversing a reachability graph, their approach is highly exponential, and does not scale efficiently with increasing number of modes. In contrast, we consider a more specialized model for resource and application modes which permits a more precise and efficient calculation of subsystem schedulability with less pessimism. In a followup paper, Phan et al. [61] simulated adaptive cruise control system by modeling it as a multi-modal system.

3.2.1 Synchronous Protocols

Synchronous protocols are characterized by higher priority of old-mode tasks, and new-mode tasks cannot start as long as there are old-mode tasks. Tindell and Alonso [76] worked with synchronous mode-change protocol. In their setting, the system wait for an idle instant after the arrival of a mode-change request. Once an idle instance is found, the system activates mode-change actions and allows new-mode tasks to release jobs. The protocol is also known as idle-time protocol. This protocol is simple and easy to implement; however, mode-change promptness is compromised due to the dependency of the first idle instant after a mode-change request. This is because of the system waiting time up to the worst-case response time of jobs if MCR co-incides with the worst job arrival sequence in the old-mode. Longer mode-change response is undesirable especially if new-mode tasks with shorter deadline are waiting for execution. Synchronous protocols are not suitable for unchanged tasks that may execute through mode-change requests uninterruptedly. Multi-modal systems proposed by Nelis et al. [54, 55] are the examples of synchronous protocols upon multiprocessor platform.

3.2.2 Asynchronous Protocols

In asynchronous protocols, new-mode tasks do not wait for old mode tasks to finish. Both new-mode and old-mode tasks can co-execute together during and after transition period. Shah et al. [67] developed an asynchronous protocol where tasks in each mode executed by Rate-Monotonic Algorithm [49]. In their settings, mode-changes are characterized by addition of new tasks, deletion of existing tasks, or changes in the parameters of tasks (e.g., increasing the sampling rate to obtain a more accurate result). The authors considered resource sharing between tasks using binary semaphore for synchronization. To avoid deadlocks while accessing critical section, the priority-ceiling protocol [67] was utilized. The mode-change protocol checks sufficient processor capacity and characteristics of priority ceiling protocol before adding a new task (the role of mode-change protocol is very similar to an admission controller) in the system. However, the authors cannot exploit the resources released by aborted tasks at the time of MCR as utility based schedulability is considered for ensuring all deadlines.

Asynchronous protocols [60, 77] allow prompt mode-changes. These set of protocols are capable of ensuring periodicity of participating tasks at the time of mode-change requests. Pedro et al. [60] argued that if there is a transient overload at the beginning of a mode-change, where the total instantaneous utilization may increase above the limit (due to new-mode jobs released immediately after mode-changes), the system will be unschedulable. For such systems, an offset is a flexible requirement provided that the system meets its overall requirements. Using offset, their proposed protocol delays newly added tasks and changed tasks from the start of a mode-change request. This reduces interferences that new tasks experience from higher priority old mode tasks; therefore, the protocol allows tasks sets to finish before deadlines. This in turn minimizes pessimism of the timing analysis, and increase the schedulability of a system. A very similar protocol by Tindell et al. [77] delays (i.e., offset) only new-mode tasks, but allows unchanged and changed tasks from old-mode to continue in new-mode maintaining inter-arrival separation constraints between jobs. Both of these approaches assume the offset is known at design time which may not be practical. Determining/optimizing offset is a difficult problem and requires significant computational effort.

3.3 Schedulability of MMS

In this section, we discuss existing schedulability analysis technique for multi-modal systems. While schedulability analysis is essential for ensuring timing integrity of any real-time system, there is no unique approach for developing such analysis due to many dimensions of the problem of schedulability in real-time systems. As a result, there exists numerous schedulability analysis for different types of multi-modal systems. Developing schedulability analysis depends mainly on tasks system, processing resource, and scheduling algorithm itself. As EDF and Fixed-Priority scheduling are the two main focuses of this thesis, following subsections present existing schedulability analysis for real-time multi-modal systems under these two categories. For EDF scheduling algorithm, typically demand-supply analysis is used, whereas response time analysis is more applicable for Fixed-Priority scheduling algorithm.

3.3.1 Demand/Supply Analysis

For demand/supply-based schedulability analysis, system designers need to ensure overall execution demand over any interval is always less than corresponding execution supply from processing resources. This type of analysis quantifies the maximum workload (demand) generated by the tasks system for any interval that needs to be completed to avoid possible deadline misses. In case of multi-modal systems, tasks sets that generate workload may change at the time of a mode-change request; therefore, the demand-supply analysis is not straightforward, and requires to consider transient workload that can be carried forward from past mode-changes. The unfinished workload from past modes is also known as carry-in, and it contributes to the instantaneous demand at the new-mode. The main difficulty with quantifying carry-in is its dependency on all previous modes processing resources. Therefore, the subroutine for determining the carry-in of a mode is complex, computationally expensive, and may require invocation of itself as a subroutine.

Stoimenov et al. [74] developed a demand/supply based schedulability analysis using real-time calculus (RTC) approach. The authors considered a single application-level mode-change. Therefore, this multi-modal system may not be usable with control systems. Phan et al. [63] performed a schedulability analysis for multi-modal systems using finite-state automaton. The authors developed equations for calculating upper bound on the workload that may come from the previous mode, and devised equations for quantifying the minimum execution (supply function) unit for a given interval. Using these two functions, authors quantified the minimum processing supply requested by each component that will ensure schedulability using finite state automaton. However, the suggested method needs to construct a tree of reachable states from the initial set of states; therefore, this method is not suitable for a system with large number of modes as the computation is highly expensive. Furthermore, the authors set constraints on the duration of how long the system can stay in a particular mode in the form of minimum and maximum mode duration. This assumption may not be suitable in control system where the systems may remain in stable states throughout the lifetime. In this thesis, we are working for developing a multi-modal real-time system taking control systems into consideration where periodic-resource [32] for each mode will serve a set of sporadic task system using only EDF.

3.3.2 Response Time Analysis

In this technique, the time to finish a job in the worst case is calculated. This finish time is known as *response-time*. In most research on fixed-priority scheduling for real-time systems, the response-time is defined iteratively (i.e., a function of response time that is calculated in the previous iteration). For ensuring the schedulability, the response time of any job must be smaller than the deadline of the corresponding task. Tindell et al. [77] developed closed recursive equation to calculate response-time of tasks after a mode-change. Real et al. [64] explored response-time analysis and evaluated the effect of offset after a mode-change to reduce the pessimism of a schedulability analysis in case of a multi-modal system.

3.4 Parallel Schedulability Analysis of MMS

Checking schedulability of multi-modal systems warrants higher computation time as a result of dependencies between modes. This dependency is due to transition between modes provisioned by a system designer. For schedulability, each mode must evaluate all valid transitions at it to accommodate the maximum carry-in it can start with. As a result, a large number of modes pose a computational challenge to existing algorithms for sequential schedulability analysis. A parallel schedulability analysis can be promising and practical alternative to traditional sequential techniques.

An efficient parallel schedulability analysis can reduce significantly the time for design-space exploration [3, 78] that may utilize schedulability tests for ensuring timing constraints while determining optimized resource parameters of a multi-mode real-time system. Schedulability analysis using parallel algorithms is a relatively unexplored area for multi-modal real-time applications. For uni-modal systems, there have been solutions with well-defined sets of conditions where each condition must pass a set of test cases. In most scenarios, the evaluation of these test case elements can be performed independently. From the perspective of parallel computing, the independent execution behavior makes the problem of uni-modal schedulability less challenging. The only non-trivial parallel schedulability analysis that we are aware of is by Feng et al. [33] and Nunna et al. [56] for uni-modal schedulability of dependent tasks represented using directed acyclic graphs (DAG). However, the schedulability analysis of multi-modal real-time systems is complex; the analysis not only depends on each mode itself, but also on the schedulability of all other modes along with mode-change sequences. Therefore, a sequential schedulability analysis may not sufficiently scale if it is used as a tool for determining optimal system parameters (e.g., hardware-resources, offset). In this thesis, we address a fundamental gap in the literature on parallel schedulability analysis suitable for design-space exploration for real-time multi-modal systems.

The performance of a parallel algorithm depends heavily on the underlying workload distribution policy. Balanced distribution of workload along with the minimal overhead due to communication/synchronization is indispensable to reduce the parallel execution time. For load balancing upon a parallel platform, different centralized [39] and distributed techniques [22] (e.g., sender/receiver-initiated [31]) may be used for distributing workload among processing nodes for checking schedulability; however, added overhead for this approach is not negligible. We propose a decentralized load-balancing technique that reduces the overhead of data distribution.

3.5 Parameters of MMS

Efficient schedulability analysis can reduce the design time significantly while finding system parameters, (also known as design-space exploration [78]), by repeated applications of the schedulability test for the problem. Design-space exploration is computationally expensive. Under real-time settings, design-space exploration has the additional penalty of checking schedulability. Faster execution time for the design-space exploration is desirable as faster execution times allow more models to be evaluated in a short time duration. This is important to reduce product time-to-market.

So far developed design-space exploration for a multi-modal system have not addressed the optimal (hardware) resource usages for modes. Phan et al. [63] for the first time worked with temporal-isolation and determined resources for each mode of a multi-modal system using the exploration of a reachability graph developed from all possible mode transitions. The algorithm may take exponential time to decide the schedulability along with the determination of resource usages. The reasons for higher computational complexity may be because of the fact that the authors combined both schedulability analysis and determination of the CPU requirements (capacity) for modes together. Although the algorithm may take exponential time to determine the resource usages of each mode, but it cannot ensure the minimum resource usages. Optimal solutions may vary due to different objective functions such as minimizing peak-temperature or minimizing the total energy consumption. In this thesis, we address the objective function of minimizing the maximum resource usages.

CHAPTER 4: MOTIVATION FOR MMS

In this chapter, we present theoretical benefits of executing periodic resource with different hardware resources. The continuous execution of a hardware resource directly influences the peak-temperature of the system; therefore, over the years, thermal-aware designs have become a prominent research issue for real-time application development. In order to avoid excessive heat generation or to reduce energy consumption, the system must utilize the processing resource prudently. Prior research work obtained resource efficiency by varying resource usages (which is analogous to change hardware mode) over time. In the following section, we show how different hardware modes can contribute to the peak system temperature. We show that higher bandwidth of a periodic resource generates a higher peak-temperature. So minimizing the maximum bandwidth of hardware-modes of a multi-modal system is an appropriate objective for minimizing peak-temperature of a periodic resource. Recently, a number of research groups have attempted to address thermal and energy constraints using control systems for processing real-time tasks [37, 36].

The development of control systems often requires the underlying system to support multiple hardware and software execution modes. Such a control system must switch between the different modes to maintain stability in a dynamic and unpredictable environment. However, each of the previously-proposed real-time control systems is *soft* real-time; that is, the system cannot guarantee that every deadline will be met, but is designed with the objective to minimize the number of deadline misses. We are unaware of a single feedback control system with hardware and software modes that is *hard* real-time, in that it guarantees that <u>no</u> deadline will be missed. The current non-existence of such a control system is due to a fundamental gap in the research literature on effective and efficient multi-modal models and schedulability analysis (i.e., analysis that determines whether a system meets all deadlines) for systems where both hardware and software may change execution modes. In this thesis, we take an initial step towards the design of such a hard real-time control system by providing a theoretical framework and associated time-efficient schedulability analysis.

In the following subsections, we develop equations to calculate the peak-temperature of a periodic resource. These equations suggest that higher bandwidth may result higher peak-temperature. This is an indication of the necessity of varying resource usages over time which can be efficiently modeled using a multi-modal system. In the appendix of this chapter, we present a brief overview of prior work on thermal-aware real-time system design.

4.1 Peak-Temperature of a System

To control the instantaneous temperature, one possible way is to utilize the processing resource efficiently. Previous work attempts to minimize the peak-temperature by opportunistic usages of CPU resources. However, erratically turning on/off the processing resources makes the thermal analysis extremely difficult. We consider a variant of periodic-resource to model the temperature of a non-continuous resource. Besides, finding the peak-temperature is difficult as both heating and cooling systems are complicated dynamic processes which depend on the surrounding environment. We could approximately model this process by applying Fourier's Law of heat conduction [23, 43, 52, 53], where thermal coefficients can be obtained by using the RC (resistor-capacitor) thermal model. Fourier's Law of heat conduction states that the rate of cooling is proportional to the difference in temperature between the object and the environment. We assume that the environment has a fixed temperature, and that temperature is scaled so that the ambient temperature is zero. Previous real-time research with thermal constraints [79, 80, 81] has also worked with the identical ambient temperature assumptions.

If we define $T(t)$ as the temperature at the time instant t, then $T(t)$ can be calculated (shown in [26]) as follows

$$T(t) = \left(\int_{t_0}^{t} (s(\tau))^{\gamma} e^{-\beta(t-\tau)} d\tau \right) + T(t_0) e^{-\beta(t-t_0)} \tag{4.1}$$

where

$$s(t) = \begin{cases} \varphi, & \text{if } t \text{ occurs during active} - \text{mode;} \\ \varphi P_{\text{off}} & \text{if } t \text{ occurs during inactive} - \text{mode;} \end{cases}$$

The function $s(t)$ can be think of as the function of time for calculating speed. The parameter β and γ are processor specific constants. Typical settings for these two is $\beta \approx 0.228, \gamma \approx 3$ (e.g., see [26]). The value of φ takes speed value using which processor would executes workload during active mode. In our setting, active processor speed is normalized to 1; therefore, the value of φ is set equal to 1. P_{off} is the fraction of speed that inactive mode executes at (compared to the speed during active mode) where $0 \leq P_{\text{off}} < 1$.

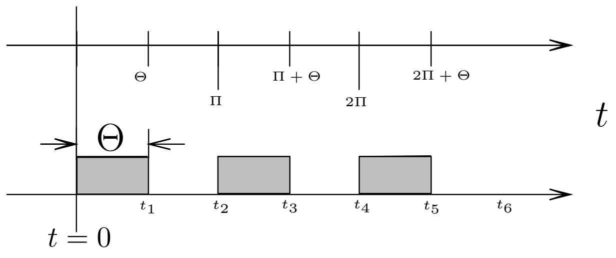

Figure 4.1: Execution pattern in thermal-aware periodic resources. Solid rectangles in the bottom portion of the figure depicting capacity each of which has a length of $\Theta^{(j)}$ time unit.

To analyze the thermal effect, we assume that the periodic resource provides <u>all</u> capacity at the beginning of the $\Pi^{(j)}$-length interval. The system is ready for execution at the time instant t_0. We assume that temperature at this time is normalized to zero and is equal to the surrounding ambient temperature. The processor will also start executing at time instant t_0. Assuming the processor starts its execution from an even numbered time instant, we define $t_{2i} = i\Pi^{(j)}$ and $t_{2i+1} = i\Pi^{(j)} + \Theta^{(j)} + \Delta^{(j)}$ for each value of $i \in \mathbb{N}$. Therefore, the gap from an even time instant to an odd time instant is $\Theta^{(j)} + \Delta^{(j)}$, whereas from an odd time instant to an even time instant - the interval length is $\Pi^{(j)} - \Theta^{(j)} - \Delta^{(j)}$. Figure 4.1 illustrates the time instants when $\Delta^{(j)}$ equals zero. The processor generates heat during its execution interval (from an even time instant to an odd time instant) and dissipates heat in the remainder of the period-of-repetition. Since the peak temperature will occur at the end of the execution in every period-of-repetition, it suffices to consider the times $t \in \{t_{2i+1} | i \geq 0\}$ as the time instants where the peak temperature may occur.

4.2 Equations for Peak-Temperature

We will use Equation 4.1 to derive the temperature at the beginning and at the ending point of each execution cycle with the assumption that the initial temperature of the processor and the ambient temperature is normalized to zero - which means $T(t_0) = 0$. Under the settings described in the previous section, we obtain the following theorem to calculate peak-temperature after a periodic interval:

Theorem 1 *Given $i \in \mathbb{N}$, the temperature of a periodic processor with the parameter $\Pi^{(j)} > 0$ and $\Theta^{(j)} \geq 0$ at the time instant t_{2i+1} can be obtained*

$$\begin{aligned} & T\left(t_{2i+1}\right) \\ & = \frac{\varphi^\gamma}{\beta}\left(1 - e^{-\beta(\Theta^{(j)}+\Delta^{(j)})}\right)\frac{1-\left(e^{-\beta\Pi^{(j)}}\right)^{i+1}}{1-e^{-\beta\Pi^{(j)}}} \\ & \quad + \tfrac{(\varphi P_{\text{off}})^\gamma}{\beta} e^{-\beta(\Theta^{(j)}+\Delta^{(j)})}\left(1 - e^{-\beta(\Pi^{(j)}-\Theta^{(j)}-\Delta^{(j)})}\right)\tfrac{1-\left(e^{-\beta\Pi^{(j)}}\right)^{i}}{1-e^{-\beta\Pi^{(j)}}} \end{aligned} \tag{4.2}$$

Proof: The proof is induction on the periodic time instants indexed by i.

(Base Case): Consider $i = 0$; at the time instant t_0, the temperature $T(t_0)$ is equal to the ambient temperature (assumed to be zero). For showing the base case, we find the temperature at t_1. Here, it is to be noted that $s(t) = \varphi$ for all $t \in [t_0, t_1)$ as the processor will be executing the workload with a constant speed in this interval. We know from the definition of t_{2i} in Section 5.1 that

$t_1 = \Theta^{(j)} + \Delta^{(j)}$ and $t_0 = 0$.

$$\begin{aligned}
&T(t_1)\\
&= \int_{t_0}^{t_1} s^{\gamma}(t) e^{-\beta(t_1 - t)} dt \ + T(t_0) e^{-\beta(t_1 - t_0)}\\
&= \int_{0}^{\Theta^{(j)}+\Delta^{(j)}} \varphi^{\gamma} e^{-\beta(\Theta^{(j)}+\Delta^{(j)}-t)} dt + 0 \cdot e^{-\beta(\Theta^{(j)}+\Delta^{(j)})}
\end{aligned}$$

$$\begin{aligned}
&= \tfrac{\varphi^{\gamma}}{\beta}\left(1 - e^{-\beta(\Theta^{(j)}+\Delta^{(j)})}\right) + 0 \cdot e^{-\beta(\Theta^{(j)}+\Delta^{(j)})}\\
&= \tfrac{\varphi^{\gamma}}{\beta}\left(1 - e^{-\beta(\Theta^{(j)}+\Delta^{(j)})}\right) \tfrac{1-(e^{-\beta\Pi^{(j)}})^{1}}{1-(e^{-\beta\Pi^{(j)}})}\\
&\quad + \tfrac{(\varphi P_{\text{off}})^{\gamma}}{\beta} e^{-\beta(\Theta^{(j)}+\Delta^{(j)})}\left(1 - e^{-\beta(\Pi^{(j)}-\Theta^{(j)}-\Delta^{(j)})}\right) \tfrac{1-(e^{-\beta\Pi^{(j)}})^{0}}{1-(e^{-\beta\Pi^{(j)}})}
\end{aligned} \tag{4.3}$$

The last step follows by noting that $1 - (e^{-\beta\Pi^{(j)}})^0$ equals zero.

(Induction Hypothesis): Assume that Equation 4.2 is true for $i = 1, 2, ...m$. Therefore

$$\begin{aligned}
&T(t_{2m+1})\\
&= \tfrac{\varphi^{\gamma}}{\beta}\left(1 - e^{-\beta\Theta^{(j)}}\right) \tfrac{1-(e^{-\beta\Pi^{(j)}})^{m+1}}{1-e^{-\beta\Pi^{(j)}}}\\
&\quad + \tfrac{(\varphi P_{\text{off}})^{\gamma}}{\beta} e^{-\beta(\Theta^{(j)}+\Delta^{(j)})}\left(1 - e^{-\beta(\Pi^{(j)}-\Theta^{(j)}-\Delta^{(j)})}\right) \tfrac{1-\left(e^{-\beta\Pi^{(j)}}\right)^{m}}{1-e^{-\beta\Pi^{(j)}}}
\end{aligned} \tag{4.4}$$

(Induction): In order to find the temperature at $t_{2(m+1)+1}$ using the Equation 4.1, we will use the temperature at $t_{2(m+1)}$ as the base temperature. From the definition, we know $t_{2m+1} = m\Pi^{(j)} + \Theta^{(j)} + \Delta^{(j)}$, $t_{2(m+1)} = (m+1)\Pi^{(j)}$ and $t_{2(m+1)+1} = (m+1)\Pi^{(j)} + \Theta^{(j)} + \Delta^{(j)}$. Temperature at

$t_{2(m+1)}$ can be obtained by the following equation:

$$\begin{aligned}
&T(t_{2(m+1)})\\
&\quad= \int\limits_{t_{2m+1}}^{t_{2(m+1)}} (\varphi P_{\text{off}})^{\gamma} e^{-\beta(t_{2(m+1)}-t)} dt + T(t_{2m+1}) e^{-\beta(t_{2(m+1)}-t_{2m+1})}\\
&\quad= \int\limits_{m\Pi^{(j)}+\Theta^{(j)}+\Delta^{(j)}}^{(m+1)\Pi^{(j)}} (\varphi \times P_{\text{off}})^{\gamma} e^{-\beta(t_{2(m+1)}-t)} dt + T(t_{2m+1}) e^{-\beta(t_{2(m+1)}-t_{2m+1})}\\
&\quad= \tfrac{(\varphi P_{\text{off}})^{\gamma}}{\beta} \left(1 - e^{-\beta(\Pi^{(j)}-\Theta^{(j)}-\Delta^{(j)})}\right)\\
&\qquad + e^{-\beta(\Pi^{(j)}-\Theta^{(j)}-\Delta^{(j)})} \times \left[\begin{array}{l} \frac{\varphi^{\gamma}}{\beta}\left(1 - e^{-\beta(\Theta^{(j)}+\Delta^{(j)})}\right) \cdot \frac{1-(e^{-\beta\Pi^{(j)}})^{m+1}}{1-e^{-\beta\Pi^{(j)}}} \\ + \frac{(\varphi P_{\text{off}})^{\gamma}}{\beta} e^{-\beta(\Theta^{(j)}+\Delta^{(j)})} \left(1 - e^{-\beta(\Pi^{(j)}-\Theta^{(j)}-\Delta^{(j)})}\right) \frac{1-\left(e^{-\beta\Pi^{(j)}}\right)^{m}}{1-e^{-\beta\Pi^{(j)}}} \end{array} \right]\\
&\quad= e^{-\beta(\Pi^{(j)}-\Theta^{(j)}-\Delta^{(j)})} \tfrac{\varphi^{\gamma}}{\beta} \left(1 - e^{-\beta\Theta^{(j)}}\right) \tfrac{1-(e^{-\beta\Pi^{(j)}})^{m+1}}{1-e^{-\beta\Pi^{(j)}}}\\
&\qquad + \tfrac{(\varphi P_{\text{off}})^{\gamma}}{\beta} \left(1 - e^{-\beta(\Pi^{(j)}-\Theta^{(j)}-\Delta^{(j)})}\right) \tfrac{1-\left(e^{-\beta\Pi^{(j)}}\right)^{m+1}}{1-e^{-\beta\Pi^{(j)}}}
\end{aligned}$$

Therefore,

$$\begin{aligned}
&T(t_{2(m+1)+1})\\
&\quad= \int\limits_{t_{2(m+1)}}^{t_{2(m+1)+1}} s^{\gamma}(t) e^{-\beta(t_{2(m+1)+1}-t)} dt + T(t_{2(m+1)}) e^{-\beta(t_{2(m+1)+1}-t_{2(m+1)})}\\
&\quad= \int\limits_{(m+1)\Pi^{(j)}}^{(m+1)\Pi^{(j)}+\Theta^{(j)}+\Delta^{(j)}} \varphi^{\gamma} e^{-\beta((m+1)\Pi^{(j)}+\Theta^{(j)}+\Delta^{(j)}-t)} dt + T(t_{2(m+1)}) e^{-\beta(\Theta^{(j)}+\Delta^{(j)})}\\
&\quad= \tfrac{\varphi^{\gamma}}{\beta} \left(1 - e^{-\beta\Theta^{(j)}}\right)\\
&\qquad + [e^{-\beta(\Pi^{(j)}-\Theta^{(j)}-\Delta^{(j)})} \tfrac{\varphi^{\gamma}}{\beta} \left(1 - e^{-\beta\Theta^{(j)}}\right) \tfrac{1-(e^{-\beta\Pi^{(j)}})^{m+1}}{1-e^{-\beta\Pi^{(j)}}}\\
&\qquad + \tfrac{(\varphi P_{\text{off}})^{\gamma}}{\beta} \left(1 - e^{-\beta(\Pi^{(j)}-\Theta^{(j)}-\Delta^{(j)})}\right) \tfrac{1-\left(e^{-\beta\Pi^{(j)}}\right)^{m+1}}{1-e^{-\beta\Pi^{(j)}}}] \times e^{-\beta(\Theta^{(j)}+\Delta^{(j)})}\\
&\quad= \tfrac{\varphi^{\gamma}}{\beta} \left(1 - e^{-\beta\Theta^{(j)}}\right)\\
&\qquad + [e^{-\beta(\Pi^{(j)}-\Theta^{(j)}-\Delta^{(j)})} \tfrac{\varphi^{\gamma}}{\beta} \left(1 - e^{-\beta\Theta^{(j)}}\right) \tfrac{1-(e^{-\beta\Pi^{(j)}})^{m+1}}{1-e^{-\beta\Pi^{(j)}}}\\
&\qquad + \tfrac{(\varphi P_{\text{off}})^{\gamma}}{\beta} \left(1 - e^{-\beta(\Pi^{(j)}-\Theta^{(j)}-\Delta^{(j)})}\right) \tfrac{1-\left(e^{-\beta\Pi^{(j)}}\right)^{m+1}}{1-e^{-\beta\Pi^{(j)}}}] \times e^{-\beta(\Theta^{(j)}+\Delta^{(j)})}
\end{aligned}$$

$$T(t_{2(m+1)+1})$$
$$= \frac{\varphi^\gamma}{\beta}\left(1 - e^{-\beta\Theta^{(j)}}\right)\frac{1-(e^{-\beta\Pi^{(j)}})^{m+2}}{1-e^{-\beta\Pi^{(j)}}}$$
$$+\frac{(\varphi P_{\text{off}})^\gamma}{\beta}e^{-\beta(\Theta^{(j)}+\Delta^{(j)})}\left(1 - e^{-\beta(\Pi^{(j)}-\Theta^{(j)}-\Delta^{(j)})}\right)\frac{1-\left(e^{-\beta\Pi^{(j)}}\right)^{m+1}}{1-e^{-\beta\Pi^{(j)}}}$$

which proves the theorem. ■

Corollary 1 *The asymptotic temperature of a* thermal-aware periodic resource *with the parameters* $\varphi > 0$, $\Pi^{(j)} > 0$, $\Delta^{(j)} \geq 0$, *and* $\Theta^{(j)} \geq 0$ *is*

$$\mathcal{T}(\Pi^{(j)}, \Theta^{(j)})$$
$$= \frac{\varphi^\gamma}{\beta}\frac{(1-e^{-\beta(\Theta^{(j)}+\Delta^{(j)})})}{(1-e^{-\beta\Pi^{(j)}})} + \frac{(\varphi\times P_{\text{off}})^\gamma}{\beta} \times e^{-\beta(\Theta^{(j)}+\Delta^{(j)})} \times \frac{1-e^{-\beta(\Pi^{(j)}-\Theta^{(j)}-\Delta^{(j)})}}{1-e^{-\beta\Pi^{(j)}}}. \tag{4.5}$$

Proof: The corollary follows directly from the Theorem 1. As the highest temperature will occur at the end of the execution in every period-of-repetition and the Equation 4.2 is non-decreasing over i, we can find the asymptotic peak-temperature by taking $\lim_{i\to\infty}$ of Equation 1, resulting in Equation 4.5; thus, the corollary is proved. ■

4.3 Multi-Modal Systems

From Equation 4.5, the free parameters, which may be conveniently tuned by a system designer are period-of-repetition $\Pi^{(j)}$ and capacity $\Theta^{(j)}$ for achieving a safe upper-bound on the temperature of a system. These two parameters contribute directly to the system temperature. The higher ratio of the capacity to the period-of-repetition implies higher peak-temperature (this ratio is frequently denoted as interface-bandwidth). To address peak-temperature, a system designer may wish to opportunistically change interface-bandwidth based on instantaneous workload utilizing some control-theoretic algorithms [28, 38]. We refer such instantaneous resource provisioning using control algorithm as control-computing. This control-computing can be very useful for ensuring predictable performance guarantee in case of unpredictable thermal computing. However, control-computing for real-time systems are not straight-forward especially which maintains hard timing constraints.

Hard real-time systems require strict accountability of the software workload, and processing capacity of the systems. Typical control computing requires adaptability to different hardware modes (e.g., defined may be using discrete interface-bandwidth) responding software mode changes (e.g., changing task systems). This changes in hardware and software (i.e., multi-modal systems) makes accountability, in turn schedulability analysis, extremely difficult. As a result, so far developed control computing supports only soft real-time systems. In this thesis, we simplify the multi-modal system schedulability using some practicable observations from control-systems (discussed in Chapter 5). To address the accountability of software and hardware for hard real-time guarantee, we observe that coupling each software mode to a hardware mode is very practical (efficient resource usages comes from software changes triggers hardware mode change, and vice versa) and makes the schedulability analysis computationally efficient. Therefore, we include these assumptions in our proposed mode-change protocol, developed schedulability analysis, and implemented algorithm for optimizing resource usages of multi-modal systems (suitable for control computing).

The support of multi-modal hard-real-time systems enables thermal-aware control computing for hard-real-time systems, which is not in scope of this dissertation. Interested readers are referred to papers [41, 42] for further exploration.

4.4 Appendix

4.4.1 Prior Work on Thermal-Aware Real-Time Systems

Chen et al.[26] explored proactive speed scheduling for periodic real-time tasks to meet both timing and thermal constraints. They solved the problem of minimizing peak-temperature in two different ways: 1. timing optimization approach, and 2. thermal-optimization approach. The authors define *converging initial temperature* as the initial temperature of a CPU in a given period where the temperature at end of the period also equals to the initial temperature. The timing optimization approach aims to minimize converging initial temperature so that the system can tolerate higher thermal constraints to complete more workload. The thermal-optimization approach targets for the minimization of the response time during a period regardless of the temperature at the beginning of the period. By experiments, the authors showed that both of these approaches outperformed reactive scheduling in minimizing peak-temperature. The authors proved that feasible speed with a converging initial temperature is a necessary condition of the schedulability under thermal constraints. Therefore, the authors first determined a feasible speed by analyzing workload that would keep the temperature unchanged; afterwards, they proved that without having a feasible speed with converging initial temperature, it was impossible to obtain a feasible speed which might keep the temperature below the user defined threshold for temperature.

In another seminal work [25], Chen and Hung worked on temperature awareness in scheduling upon uniprocessor or homogeneous multiprocessors (Single-Chip Multiprocessor, Multi-Chip Multiprocessor) using dynamic voltage scaling where the cooling process was modeled by Fourier's law. For multiprocessor systems, the authors evaluated different approaches proposed for partitioned scheduling, including the first-fit, the next-fit, the best-fit, and the worst-fit algorithms (see Johnson's thesis [45] for more about near optimal bin packing algorithms). The authors determined an approximation ratio for the maximum temperature for both homogeneous and heterogeneous multiprocessor. They showed in the paper that Largest Task First (LTF) strategy performed better for minimizing peak-temperature and could have a 1.13 approximation upon uniprocessor, 3.072 for homogeneous and 6.444 for multiprocessor. Using analytical analysis, the authors concluded that LTF's better performance came from the fact that it derived solutions with more balanced load distribution. From the perspective of temperature minimization, this paper has a shortcoming; this paper minimizes temperature as second objective. First objective was to minimize the energy consumption.

Chantem et al. [24] made very interesting observations for maximizing workload under thermal constraints. While working with proactive scheduling, the authors found a means for maximizing the work completed for processors with discrete speed levels and non-negligible transition overheads. The authors determined a work conserving speed schedule such that the peak temperature constraints were met and the total work completed was maximized using a dynamic voltage scaling (DVS) control policy for processors with discrete speed levels. The authors also showed that a schedule that would complete the maximum amount of work must be a periodic speed schedule. For obtaining this result, the authors first proved that a schedule that completed the maximum amount of work would allow the chip temperature to reach the highest temperature at the end of the application of highest speed in the sequence. But the authors did not provide any formula for quantifying highest temperature even with basic speed settings. This thesis fills this gap, and develops a formula for calculating the upper bound using approximation ratio on peak temperature that can be reached from given workload with the guarantees of meeting all deadline constraints. The problem considered in this thesis is orthogonal to Chantems's work in a sense that Chantem works for maximizing workload under thermal constraints whereas this thesis minimizes the peak-temperature for a given workload.

Wang et al. [79, 80, 81] studied schedulability analysis under the reactive setting. The authors worked with a very simple idea: the processor runs at the highest speed when there is backlogged workload and the temperature is below the threshold. Otherwise, following speed scaling actions will be taken: Whenever the backlogged workload is empty, the processor idles; Whenever the temperature hits threshold, the processing speed is reduced (through DVS or appropriate clock throttling) to an equilibrium speed (denoted by S_E) that keeps the temperature constant. Authors provided formula for calculating equilibrium speed. In a very similar publication [79], the author obtained a closed-form delay formula for the leaky bucket task arrival model and showed this simple reactive speed control decreased the response time of tasks compared with any constant-speed scheme. The authors argued that these techniques can easily be implemented using the thermal management facilities on many currently available microprocessors. Reactive speed scaling has the inherent disadvantage of having the system running above the threshold for quite some time at the vicinity of the transitions to equilibrium speed.

The previous work on proactive and reactive scheme assumed both simple task models and the existence of “ideal” processor speeds which may not be available even for the recent top-of-the-line microprocessors. In this chapter, we developed equation to calculate peak-temperature of a proactive scheduler. We remove some ideal assumptions by working with only two discrete speed modes and the more general sporadic task model. Furthermore, we account for the transition overhead due to switching between modes.

CHAPTER 5: MODE CHANGE PROTOCOL OF MMS

Previous research on real-time multi-mode systems has assumed that the system is executing upon a dedicated processing platform. Recently researchers have started to address the problem of guaranteeing hard deadlines of temporally-isolated subsystems in multi-modal systems. However, most of this recent research does not have full support for both resource-level mode changes and application-level mode changes does not exist. The lack of support for both resource and application mode changes severely limits the ability of the subsystem to adapt to dynamic internal and external events. In this thesis, we address this drawback by providing a multi-modal system for application executing in a temporally-isolated environment under both resource and application-level mode changes.

In this chapter, we propose a mode-change protocol taking control systems into account. A viable protocol for changing modes facilitates the development of real-time control systems. Adaptive control systems possess the ability of maintaining stability even in dynamic and unpredictable operating environments. Control systems switch between modes observing environmental variables (e.g., temperature). Each mode may consist of different tasks with different QoS levels. In the case of a real-time task set, a system designer needs to ensure all timing constraints are met for the selected task set during operation. For a hard real-time task set, extra attention is required during the transition period as the system may get overloaded due to jobs from both old mode and new mode tasks. Therefore, an effective mode change protocol with schedulability test is essential in developing real-time control computing systems.

5.1 Proposed Mode-Change Protocol

To exploit continuously improved hardware capability, a real-time system may consist of multiple real-time subsystems (also known as compositional systems). A subsystem may have may have their own local scheduler different from the system (upper) level scheduler. To be schedulable, all subsystems must specify their worst case requirements carefully so that top level scheduler can determine budget for each subsystem. For this thesis, we exclusively focus on the subsystem-level schedulability.

For each subsystem, we consider real-time application workload (software) and the processing resource (hardware) to have multiple modes. We denote $\tau \overset{\text{def}}{=} \{\tau^{(i)} | 1 \le i \le q\}$ as the set of all software modes. The real-time workload of $\tau^{(i)}$ is modeled by the *sporadic tasks model* [51]. In order to ensure temporal isolation and also hard deadlines, we explicitly couple each software mode $\tau^{(i)}$ to a hardware mode $\Omega^{(i)}$, and constitute a subsystem mode $M^{(i)}$, where $i \in \{1, \dots, q\}$. The processing resource $\Omega^{(i)}$ is modeled by the *explicit-deadline periodic (EDP) resource model* [32]. Throughout this thesis, we assume that timing parameters are natural numbers. This assumption may not be restrictive as all timing parameters can be expressed in terms of the number of clock ticks.

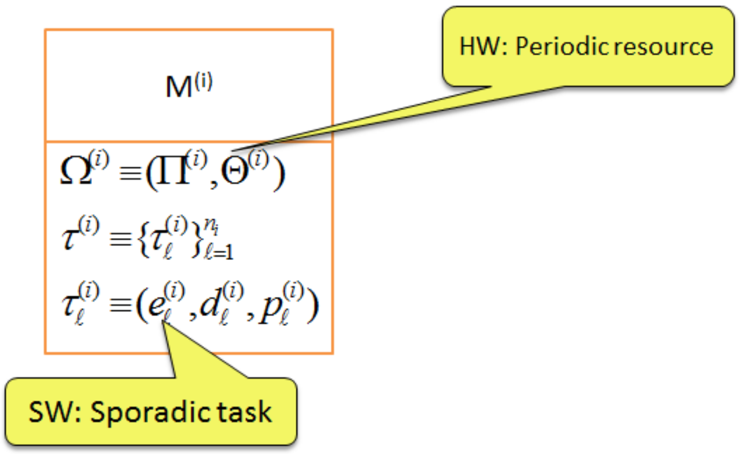
HW: Periodic resource
$M^{(i)}$
$\Omega^{(i)} \equiv (\Pi^{(i)}, \Theta^{(i)})$
$\tau^{(i)} \equiv \{\tau_\ell^{(i)}\}_{\ell=1}^{n_i}$
$\tau_\ell^{(i)} \equiv (e_\ell^{(i)}, d_\ell^{(i)}, p_\ell^{(i)})$
SW: Sporadic task

Figure 5.1: Components of a mode.

5.1.1 Modes

We consider each subsystem mode to be specified by a three-tuple $\left(\tau^{(i)}, \Omega^{(i)}, N^{(i)}\right)$ which respectively characterizes the real-time workload generated by a sporadic task system, the minimum processor execution guaranteed by an EDP resource, and the minimum mode duration in terms of "number of resource periods" $N^{(i)}$. The interpretation of $N^{(i)}$ is that the subsystem remains in mode $M^{(i)}$ for at least $N^{(i)} \cdot \Pi^{(i)}$ time units. If $N^{(i)}$ equals zero, then there could be a new mode change request as soon as it enters in the new modes. Having $N^{(i)}$ equals to zero may be problematic especially for control systems that looks for stability while ensuring hard real-time constraints ($N^{(i)}$ equals zero will allow a control system to quickly change modes in a short period of time where each new mode may release their own workload, but the allocated hardware resource in a mode may not get enough time to handle this cumulative workload in time). In addition, changing hardware modes of a processing resource using DVFS may not be instantaneous which can be easily model based on the requirements using positive $N^{(i)}$.

5.2 Mode-Change Request (MCR) Model

At runtime, the subsystem switches between modes during a sequence mcr_0, mcr_1, mcr_2, $\ldots$ of mode-change requests. The k'th mode-change request mcr_k (for $k > 0$) is characterized by a three-tuple $(M^{(i)}, M^{(j)}, t_k)$ where t_k represents the *transition time*, $M^{(i)}$ is the *old mode* executing prior to t_k, and $M^{(j)}$ is the *new mode* executing after t_k (where $M^{(i)} \neq M^{(j)}$ and $i, j \in \{1, \ldots q\}$). We assume that if $i < j$, then mcr_i occurs prior to mcr_j (i.e., $t_i \leq t_j$); that is, the mode-change requests are indexed in ascending-time order. Mode-change request $\mathsf{mcr}_0 \stackrel{\text{def}}{=} (M^{(0)}, \cdot, 0)$ represents transition from the null-mode $M^{(0)}$ to any mode in $\{M^{(1)}, \ldots, M^{(q)}\}$ at subsystem-start time (assumed to be zero). After $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$ has been issued at time t_k, there may be a transition period during which jobs generated by $M^{(i)}$ have not completed and $M^{(j)}$ has not yet begun to generate jobs.

In order to facilitate quick changes of modes, the system designer may allow a transition

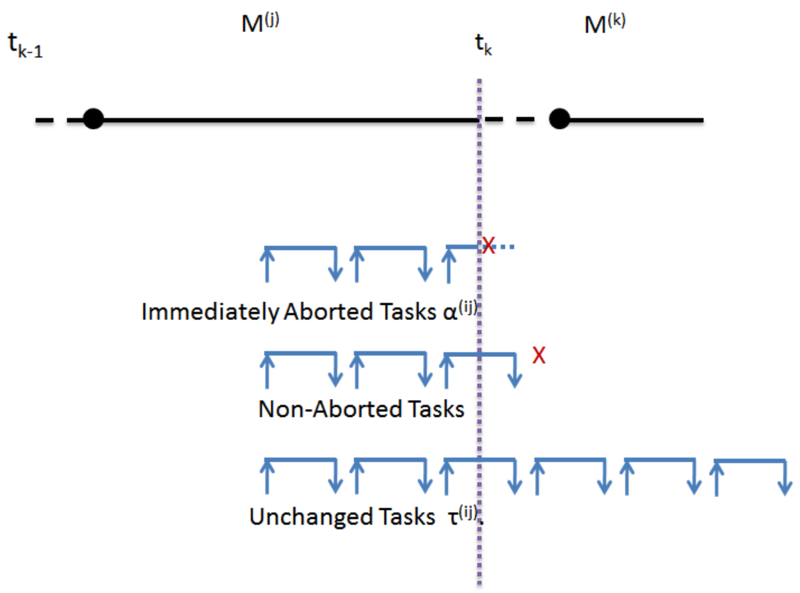
t_{k-1}
$M^{(j)}$
t_k
$M^{(k)}$
Immediately Aborted Tasks $\alpha^{(ij)}$
Non-Aborted Tasks
Unchanged Tasks $\tau^{(ij)}$

Figure 5.2: Different types of tasks.

period, called the *offset*. For any $M^{(i)}, M^{(j)} \in \{M^{(1)}, \ldots, M^{(q)}\}$, we denote the length of the transition period (called the *offset*) by $\delta^{(ij)}$. (Note, if there is no transition period, we set $\delta^{(ij)} = 0$.) During the transition period after mcr_k, only (non-aborted) jobs of $\tau^{(i)}$ are permitted to execute. At and after time $t_k + \delta^{(ij)}$, the task system $\tau^{(j)}$ may generate and execute jobs along with any remaining execution of jobs from $\tau^{(i)}$. While some jobs from $M^{(i)}$ may continue to execute in the transition period after t_k, the subsystem designer may choose to abort some jobs. We denote by $\alpha^{(ij)}(\subseteq \tau^{(i)})$ the set of tasks of $\tau^{(i)}$ which abort non-completed jobs at the transition time (e.g., t_k) for any mode change from $M^{(i)}$ to $M^{(j)}$. The subsystem designer may want some tasks that are common to both mode $M^{(i)}$ and $M^{(j)}$ to be unaffected by the mode change request mcr_k. We denote these unchanged tasks by $\tau^{(ij)}(\subseteq \tau^{(i)} \cap \tau^{(j)})$. At last, there may be some tasks that are common in both modes, but have some properties changed, we treat these tasks as finished tasks in the old mode. During the transition period after mcr_k, the EDP resource may also change execution behavior. We denote the resource parameters for the transition period between $M^{(i)}$ and $M^{(j)}$ by $\Omega^{(ij)} \stackrel{\text{def}}{=} (\Pi^{(ij)}, \Theta^{(ij)}, \Delta^{(ij)})$. We assume that the offset $\delta^{(ij)}$ is some multiple of $\Pi^{(ij)}$. Figure 5.3 illustrates a possible resource execution pattern between successive mode changes.

Given the above definitions, we may observe four phases with respect to mode-change request $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$ (and the previous request $\mathsf{mcr}_{k-1} = (M^{(h)}, M^{(i)}, t_{k-1})$):

1. $[t_{k-1} + \delta^{(hi)}, t_k)$: jobs of $\tau^{(i)}$ are executed upon $\Omega^{(i)}$;
2. $[t_k, t_k + \delta^{(ij)})$: non-aborted jobs ($\tau^{(i)} \setminus \alpha^{(ij)}$) with remaining execution at t_k execute upon $\Omega^{(ij)}$;
3. $[t_k + \delta^{(ij)}, t_{k+1})$: Incompleted non-aborted jobs ($\tau^{(i)} \setminus \alpha^{(ij)}$) at $t_k + \delta^{(ij)}$ and jobs of $\tau^{(j)}$ execute upon $\Omega^{(j)}$;
4. $[t_{k-1} + \delta^{(hi)}, t_{k+1})$: unchanged tasks ($\tau^{(ij)}$) act independent of mode change request.

The above task classifications follow the taxonomy found in the real-time mode-change survey by Real and Crespo [64].

In a general subsystem, the interval of separation between successive mode-change requests may be determined by upper and lower bounds on the amount of time that a subsystem may execute in a given mode (e.g., the multi-mode abstraction proposed by Phan et al. [63]). In this thesis, we restrict the mode-change requests and transition intervals to occur only at period boundaries in the EDP model and drop the specification of an upper bound on the separation of mode-change requests. That is, for any mode-change request $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$, $\delta^{(ij)}$ must be a multiple of $\Pi^{(ij)}$. Furthermore, for any two successive mode-change requests $\mathsf{mcr}_{k-1} = (M^{(h)}, M^{(i)}, t_{k-1})$ and $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$, we require that

$$t_k = t_{k-1} + \delta^{(hi)} + a\Pi^{(i)} \tag{5.1}$$

for some $a \in \mathbb{N}^+$ where $a \geq N^{(i)}$. The assumption that mode-change request occur at period boundaries is precisely valid in control systems such as energy-aware or thermal-aware systems where there might be DPM techniques that inactivate the processing resource in periodic intervals. For control systems, the mode change may occur precisely at the periodic sampling/actuation boundaries (Hettiarachchi et al. [42]). We also assume that a non-aborted job may span no more than one mode-change request; i.e.,

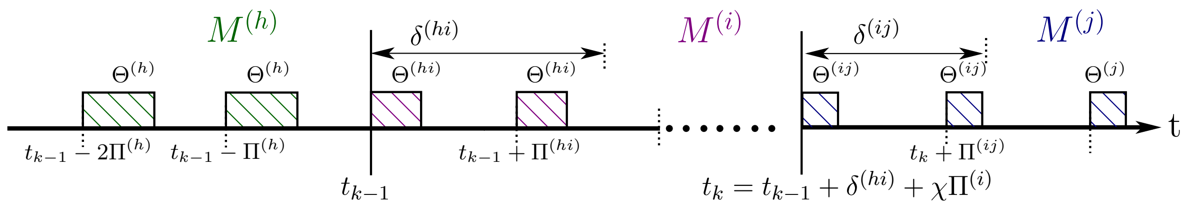

Figure 5.3: Execution pattern of a multi-modal real-time subsystem. The shaded areas indicate times during which tasks of each mode execute on the processor.

$$t_{k+1} - t_k \geq \max(d_{\max}^{(h)}, d_{\max}^{(hi)}). \tag{5.2}$$

where $d_{\max}^{(h)} \overset{\text{def}}{=} \max_{\tau_\ell^{(h)} \in \tau^{(h)} \setminus \alpha^{(hi)}} \left\{ d_\ell^{(h)} \right\}$ represents the maximum (non-aborted) job deadline from mode $M^{(h)}$ to $M^{(i)}$ and $d_{\max}^{(hi)} \overset{\text{def}}{=} \max_{\tau_\ell^{(h)} \in \tau^{(hi)}} \left\{ d_\ell^{(hi)} \right\}$ represents the maximum (unchanged) job deadline from $M^{(h)}$ to $M^{(i)}$. To the best of our knowledge, all known real-time mode-change protocols implicitly or explicitly require this constraint.

Although unfinished tasks would not span more than one mode change request, these tasks along with aborted tasks can leave transient effect in case of successive mode-change requests. If there are no cascading mode-changes and transient effect from carry-in, then the schedulability analysis becomes easier (identical to unimodal systems) and converges very quickly. We are interested in the worst case effect rather than the better one. Given the possibility of cascading mode-change, the worst case is going to be how many times mode-changes need to be evaluated before this transient effect disappears. We utilize an iterative (instead of recursive exploration of reachability graph [63]) approach and develop a pseudo-polynomial asymptotic upper bound on multi-modal schedulability analysis in the next chapter.

CHAPTER 6: EDF SCHEDULABILITY OF MMS

EDF schedulability analysis requires quantification of the maximum workload (demand) in an interval that needs to be completed for checking possible deadline misses. In case of multi-modal systems, a task set that generates workload cannot be assumed fixed for any interval due to arbitrary mode switching. Besides, transient workload that can be carried forward from the previous mode (also known as carry-in) contributes to the instantaneous demand at the new mode. The main difficulty with quantifying carry-in is due to its dependency on application workload and processing capacity (supply) of all previously executed modes. Therefore, the method for determining carry-in is complex and often requires invoking itself as a recursive subroutine. Phan et al. [63] modeled the problem as finite state automaton, and constructed a reachability graph for determining the minimum resource requirements. An obvious disadvantage of this method is time-complexity which is exponential. Stoimenov et al. [74] also worked with EDF-schedulability analysis for a very simplistic model which may not be suitable for control systems. In this chapter, we develop a schedulability analysis technique for multi-modal real-time systems where modes and associated mode changing protocol are chosen taking control systems into account.

We first derive a sufficient schedulability analysis for a setting where the sequence of mode changes is a priori known and each mode has its workload scheduled by the earliest-deadline-first (EDF) scheduling algorithm. We refer this setting as *concrete* mode-change request setting. While concrete sequences of mode changes are unlikely to be present in most multi-mode and control systems, schedulability analysis developed for such sequences can be extended (as discussed in Section 6.2) to the more practical scenario of *non-concrete* sequences of mode change requests (i.e., sequences in which the mode-changes requests are not a priori known).

6.1 Concrete Mode Changes

In this section, we consider the following problem:

> $\mathsf{Concrete\text{-}MM\text{-}Sched\ Problem}$: Given modes $M^{(1)}, \ldots, M^{(q)}$, resources $\Omega^{(ij)}$, offset $\delta^{(ij)}$, unchanged tasks $\tau^{(ij)}$, and aborted tasks $\alpha^{(ij)}$ for all $i, j \in \{1, \ldots, q\}$ $(i \neq j)$, and concrete sequence of mode-change request $\mathsf{mcr}_0, \mathsf{mcr}_1, \mathsf{mcr}_2, \ldots$ that satisfies Equations 5.1 and 5.2, determine whether all jobs (under all legal job arrival sequence) are EDF-schedulable (i.e., EDF always meets each job's deadline).

6.1.1 Definitions

A major challenge for schedulability analysis of multi-mode subsystems (over uni-mode systems) is dealing with the execution of non-aborted jobs from the old mode while a new mode is executing. If all tasks abort jobs at the mode-change request, then the analysis would be identical to traditional uni-mode schedulability analysis. However, aborting jobs are not always appropriate, especially if aborting jobs may leave the subsystem in an unstable state. Thus, to be able to accurately determine the schedulability of multi-mode subsystems, we must precisely quantify the workload and demand that may *carry-in* from the old mode to the new mode for a mode-change request $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$. The following definitions with respect to mcr_k are useful in quantifying this workload. The definitions are with respect to a concrete sequence of mode-change requests mcr_0, mcr_1, …, mcr_{k-1}, $\mathsf{mcr}_k, \ldots$. We will make use of an indicator function $\mu_{\geq 0}(x)$ which is zero if $x < 0$ and is one otherwise; we will also use the notation $(x)_+ \stackrel{\text{def}}{=} \max(0, x)$.

Definition 5 (Carry-In Execution for mcr_k) *The* carry-in execution *for mode-change request mcr_k is the maximum remaining execution of non-aborted jobs from mode $M^{(i)}$ for tasks $\tau^{(i)} \setminus \{\tau^{(ij)} \cup \alpha^{(ij)}\}$ at time $t_k + \delta^{(ij)}$ that arrive prior to t_k and maximum remaining execution of unchanged tasks (i.e., $\tau^{(ij)}$) that have arrival before $t_k + \delta^{(ij)}$. We denote this value by $\mathsf{ci}(\mathsf{mcr}_k)$.*

Definition 6 (Mode-Change DBF for mcr_k) *For* $\mathsf{mcr}_{k-1} = (M^{(h)}, M^{(i)}, t_{k-1})$ *and* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$*, and* $x, \phi \in \mathbb{R}_{\geq 0}$*, the* mode-change demand function *for any* mcr_k *is the maximum total execution demand of jobs of* $\tau^{(i)}$ *in the interval* $[t_k - x, t_k + \phi]$ *(and any carry-in jobs when* $t_k - x$ *corresponds to the end of a transition for* mcr_{k-1} *– i.e.,* $t_k - x = t_{k-1} + \delta^{(hi)}$*). For a task* $\tau_\ell^{(i)} \in \tau^{(i)}$*, we denote its contribution to the total demand by* $\mathsf{mcdbf}(\mathsf{mcr}_k, \tau_\ell^{(i)}, x, \phi)$*. The total demand of all jobs for the mode change is denoted by* $\mathsf{mcdbf}(\mathsf{mcr}_k, x, \phi)$.

In other words, to be included in the demand, the jobs of finished or aborted tasks must arrive in the interval $[t_k - x, t_k)$ and have deadline in the interval $[t_k - x, t_k + \phi]$. For unchanged tasks of $\tau^{(ij)}$, the mode-change demand function includes jobs that have an arrival and deadline in the interval $[t_k - x, t_k + \phi]$ (i.e., we permit unchanged tasks that arrive after t_k to be included in the execution demand). The mode-change demand also includes the entire execution requirements of all non-aborted jobs generated in $[t_k - x, t_k)$ that have deadlines prior to $t_k + \phi$, if $\tau_\ell^{(i)} \in \tau^{(i)} \setminus \alpha^{(ij)}$; otherwise, the demand includes the execution of non-aborted jobs that arrive and have deadline in $[t_k - x, t_k]$ and only the possible completed portion of aborted jobs that arrive in $[t_k - x, t_k)$ and have deadlines in $(t_k, t_k + \phi]$, if $\tau_\ell^{(i)} \in \alpha^{(ij)}$ (see Figure 6.6). If x equals $t_k - t_{k-1} - \delta^{(hi)}$, then the total demand also includes the carry-in execution for mcr_{k-1} $\{$i.e., $\mathsf{ci}(\mathsf{mcr}_{k-1})\}$. The demand $\mathsf{mcdbf}(\mathsf{mcr}_k, x, \phi)$ may be computed by

$$\textstyle\sum_{\tau_\ell^{(i)} \in \tau^{(i)}} \mathsf{mcdbf}(\mathsf{mcr}_k, \tau_\ell^{(i)}, x, \phi) + \mu_{\geq 0}(x - (t_k - t_{k-1} - \delta^{(hi)})) \cdot \mathsf{ci}(\mathsf{mcr}_{k-1}). \tag{6.1}$$

Definition 7 (Carry-In DBF) *The* carry-in demand-bound function *for mode-change request* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$ *and* $\phi \in \mathbb{R}_{\geq 0}$ *is the maximum remaining execution of jobs of tasks* $\tau^{(i)} \setminus \alpha^{(ij)}$ *that arrive prior to* t_k *(or prior to* $t_k + \delta^{(ij)}$ *for* $\tau^{(ij)}$ *tasks) and have deadline in the interval* $[t_k, t_k + \phi]$ *for any* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$*. We denote this quantity by* $\mathsf{cidbf}(\mathsf{mcr}_k, \phi)$*.*

In the next three definitions, we define the minimum resource-execution supply function for three different scenarios: 1) before an MCR; 2) during the transition; and 3) after the MCR transition.

Definition 8 (Pre-Mode-Change SBF) *The mode-change supply-bound function, prior to any mode-change request* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$*, is the minimum execution guaranteed by* $\Omega^{(i)}$ *over the interval* $[t_k - x, t_k]$*. We denote this service by* $\beta^i_{\mathsf{prior}}(x)$*.*

Definition 9 (Transition-Mode-Change SBF) *The mode-change supply-bound function, during the transition period mode-change request* mcr_k*, is the minimum execution guaranteed by* $\Omega^{(ij)}$ *and* $\Omega^{(j)}$ *over the interval* $[t_k, t_k + \phi]$*. We denote this service by* $\beta^{i,j}_{\mathsf{trans}}(\phi)$*.*

Definition 10 (Post-Mode-Change SBF) *The mode supply-bound function, following any mode-change request* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$*, is the minimum execution guaranteed by* $\Omega^{(ij)}$ *to carry-in jobs of* $M^{(i)}$ *and by* $\Omega^{(j)}$ *to* $M^{(j)}$ *(and any carry-in jobs) over the interval* $[t_k + \delta^{(ij)} - x, t_k + \delta^{(ij)} + y]$ *(for* $0 \leq x \leq \delta^{(ij)}$*). We denote this service by* $\beta^{i,j}_{\mathsf{post}}(x, y)$*.*

6.1.2 Deriving MCR Service-Bound Function

In this subsection, we derive lower bounds on the supply functions of Definitions 8, 9, and 10. We start with a lower bound for $\beta^i_{\mathsf{prior}}(x)$.

Lemma 1 *For* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$ *and* $x \geq 0$*,*

$$\beta^i_{\mathsf{prior}}(x) \geq \ a\Theta^{(i)} + \min\Big(\Theta^{(i)}, \big(x - \big((a+1)\Pi^{(i)} - \Theta^{(i)}\big)\big)_+\Big) \tag{6.2}$$

where $a \overset{def}{=} \left\lfloor \frac{x}{\Pi^{(i)}} \right\rfloor$*.*

Proof: By Equation 5.1, t_k will coincide with the end of a resource period. The minimum execution over an interval beginning at t_k and extending to the left for x time units when the execution is provided at the beginning of every resource period. Figure 6.1 depicts this scenario. Therefore, for any interval of length x, at least $a \overset{\text{def}}{=} \left\lfloor \frac{x}{\Pi^{(i)}} \right\rfloor$ capacities for a full resource periods follows $\min\Big(\Theta^{(i)}, \big(x - \big((a+1)\Pi^{(i)} - \Theta^{(i)}\big)\big)_+\Big)$ (forward diagonally shaded region in Figure 6.1). ■

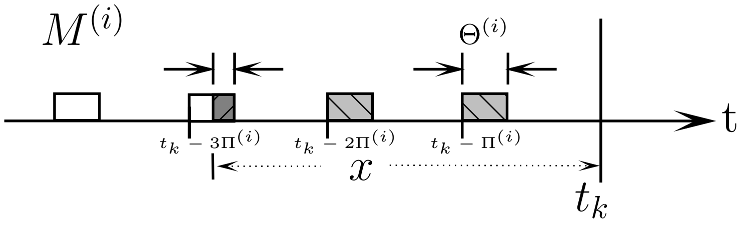

Figure 6.1: Minimum supply in x before mcr_k.

The following corollary may be immediately obtained by taking a linear lower-bound of right-hand side of Equation 6.2. (A similar lower bound is due to Shin and Lee [70].)

Corollary 2 *For* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$ *and* $x \geq 0$*,*

$$\beta^i_{\mathsf{prior}}(x) \geq x \cdot \frac{\Theta^{(i)}}{\Pi^{(i)}} - \frac{\Theta^{(i)}}{\Pi^{(i)}} \left(\Pi^{(i)} - \Theta^{(i)}\right). \tag{6.3}$$

The next two lemmas give bounds for the transition and post mode-change supply-bound functions. The proofs of these lemmas are similar to Lemma 1.

Lemma 2 *For* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$*,* $y \geq 0$*, and* $x : 0 \leq x \leq \delta^{(ij)}$*,*

$$\begin{aligned} \beta^{i,j}_{\mathsf{post}}(x, y) \quad &\geq a\Theta^{(ij)} + \min\left(\Theta^{(ij)}, \left(x - \left((a+1)\Pi^{(ij)} - \Theta^{(ij)}\right)\right)_+\right) \\ &\quad + f\Theta^{(j)} + \min\left(\Theta^{(j)}, \left(y - \left(f\Pi^{(j)} + \Delta^{(j)} - \Theta^{(j)}\right)\right)_+\right) \end{aligned} \tag{6.4}$$

where $a \overset{def}{=} \left\lfloor \frac{x}{\Pi^{(ij)}} \right\rfloor$ *and* $f \overset{def}{=} \left\lfloor \frac{y}{\Pi^{(j)}} \right\rfloor$*.*

Proof: We will quantify the minimum supply for $[t_k + \delta^{(ij)} - x, t_k + \delta^{(ij)} + y]$ by determining the minimum supply for the intervals $[t_k + \delta^{(ij)} - x, t_k + \delta^{(ij)}]$ and $[t_k + \delta^{(ij)}, t_k + \delta^{(ij)} + y]$ separately. First, we consider $[t_k + \delta^{(ij)} - x, t_k + \delta^{(ij)}]$. By Equation 5.1, t_k will coincides with the end of a resource period; furthermore, $\delta^{(ij)}$ is a multiple of $\Pi^{(ij)}$. Thus, $t_k + \delta^{(ij)}$ also coincides with the end of a resource period. The minimum execution over an interval beginning at $t_k + \delta$ and extending to the left for x time units when the execution is provided at the beginning of every resource period. Figure 6.1 depicts this scenario for $\Omega^{(i)}$; the same scenario holds for $\Omega^{(ij)}$. Therefore, for any interval of length x, at least $a \overset{\text{def}}{=} \lfloor \frac{x}{\Pi^{(ij)}} \rfloor$ capacities for a full resource periods follows $\min\left(\Theta^{(ij)}, \left(x - \left((a+1)\Pi^{(ij)} - \Theta^{(ij)}\right)\right)_+\right)$ (forward diagonally shaded region in Figure 6.1).

Now consider the interval $[t_k + \delta^{(ij)}, t_k + \delta^{(ij)} + y]$. The minimum supply for an interval that starts from $t_k + \delta^{(ij)}$ will occur if the resource is available as late as possible. The full resource capacity must be supplied within Δ_j after the start of any period-of-repetition. Figure 6.2 is showing the sequence that produce minimum supply for an interval of length x where $t_s \overset{\text{def}}{=} t_k + \delta^{ij}$. Therefore, for any interval length of x, at least $f \overset{\text{def}}{=} \lfloor \frac{y}{\Pi^{(j)}} \rfloor$ capacities for f full period-of-repetitions are followed by $\min\left(\Theta^{(j)}, \left(y - \left(f\Pi^{(j)} + \Delta^{(j)} - \Theta^{(j)}\right)\right)_+\right)$.

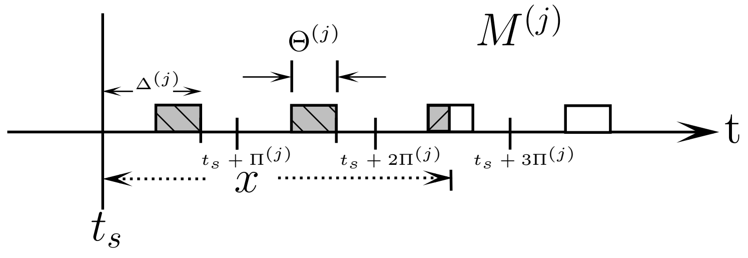

Figure 6.2: The minimum supply during transition.

■

Corollary 3 *For* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$ *and* $x \geq 0$*,*

$$\begin{aligned} \beta^{i,j}_{\mathsf{post}}(x,y) \geq\ & x \cdot \frac{\Theta^{(ij)}}{\Pi^{(ij)}} - \frac{\Theta^{(ij)}}{\Pi^{(ij)}}\left(\Pi^{(ij)} - \Theta^{(ij)}\right) \\ & + y \cdot \frac{\Theta^{(i)}}{\Pi^{(i)}} - \frac{\Theta^{(i)}}{\Pi^{(i)}}\left(\Delta^{(i)} - \Theta^{(i)}\right). \end{aligned} \tag{6.5}$$

Lemma 3 *For* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$ *and* $\phi \geq 0$*,*

$$\begin{aligned} \beta^{i,j}_{\mathsf{trans}}(\phi) \geq\ & b\Theta^{(ij)} + \min\left(\Theta^{(ij)}, \left(\min(\phi, \delta^{(ij)}) - \left(b\Pi^{(ij)} + \Delta^{(ij)} - \Theta^{(ij)}\right)\right)_+\right) \\ & + d\Theta^{(j)} + \min\left(\Theta^{(j)}, \left((\phi - \delta^{(ij)})_+ - \left(d\Pi^{(j)} + \Delta^{(j)} - \Theta^{(j)}\right)\right)_+\right) \end{aligned} \tag{6.6}$$

where $b \overset{def}{=} \left\lfloor \frac{\min(\phi, \delta^{(ij)})}{\Pi^{(ij)}} \right\rfloor$ *and* $d \overset{def}{=} \left\lfloor \frac{(\phi - \delta^{(ij)})_+}{\Pi^{(j)}} \right\rfloor$.

Proof: We omit the proof for this Lemma as it is nearly identical to Lemma 2.

■

6.1.3 Deriving the Mode-Change DBF

In this section, we derive upper bounds on the demand function of Definition 6 for the different types of tasks present for mode change $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$ (i.e., finished, unchanged, and aborted tasks). We first derive an upper bound for the finished tasks (tasks that will not be continued in next mode, but the last job is allowed to finish its execution). The bounds obtained in this subsection are similar to general results of Phan et al. [63]; however, our results are more specific to the sporadic task and periodic resource models permitting a more precise analysis of carry-in in later subsections. Furthermore, Phan et al. [63] do not consider aborted jobs in their analysis.

Lemma 4 *For* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$, $\tau_\ell^{(i)} \in \tau^{(i)} \setminus \{\tau^{(ij)} \cup \alpha^{(ij)}\}$, *and* $x, \phi \geq 0$, *if* $d_\ell^{(i)} > \phi$, *then* $\mathsf{mcdbf}\left(\mathsf{mcr}_k, \tau_\ell^{(i)}, x, \phi\right)$ *is maximized by job arrival sequence where the last job of* $\tau_\ell^{(i)}$ *arrives at* $t_k + \phi - d_\ell^{(i)}$ *and previous jobs arrive as late as legally allowed.*

Proof: Let J be the set of jobs that arrive according to the sequence described in the lemma with arrivals in $[t_k - x, t_k)$ and deadlines in $[t_k - x, t_k + \phi]$. Assume a different sequence of jobs J' of $\tau^{(i)}$ other than J maximizes the demand over the interval. We will show by induction over the jobs of J' that we may transform J' into J without decreasing the total demand. We denote the sequence of jobs (in decreasing order of arrival) as $j_0, j_1, \ldots$ for J and $j'_0, j'_1, \ldots$ for J'. Let j'_k denote the latest arriving job of sequence J' that does have the same arrival-time as j_k in J (i.e, j_i and j'_i arrive at identical times for $i = 0, 1, \ldots, k-1$). As j_k arrives as late as legally possible, it must be that j'_k has an earlier arrival time than j_k. Since j_k has both deadline and arrival in $[t_k - x, t_k + \phi]$, moving j'_k's arrival time to correspond to j_k will ensure that j'_k is still in the interval and does not violate the minimum interval constraint for $\tau_\ell^{(i)}$. Let's call this new sequence with j'_k moved to the corresponding j_k arrival as J''. It is clear that the demand of J'' does not decrease when compared to J'. By repeated application of this transformation, we may change J' to J without ever decreasing the demand which implies that J also maximizes demand. Figure 6.3 depicts the sequence corresponding to J.

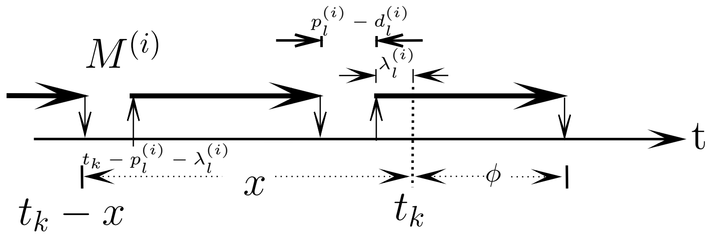

Figure 6.3: Jobs ($d_l^{(i)} > \phi$) that maximizes mcdbf.

■

The next lemma obtains the maximum requests from finished tasks with $d_\ell^{(i)} \leq \phi$ which can be shown via a similar technique to Lemma 4. Figure 6.4 is depicting the job arrival sequence which would maximizes mode change demand for a given x and ϕ.

Lemma 5 *For* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$*,* $\tau_\ell^{(i)} \in \tau^{(i)} \setminus \{\tau^{(ij)} \cup \alpha^{(ij)}\}$*, and* $x, \phi \geq 0$*, if* $d_\ell^{(i)} \leq \phi$*, then* $\mathsf{mcdbf}\left(\mathsf{mcr}_k, \tau_\ell^{(i)}, x, \phi\right)$ *is maximized by job arrival sequence where the last job of* $\tau_\ell^{(i)}$ *generated in* $[t_k - x, t_k)$ *occurs an arbitrarily small* $\epsilon > 0$ *prior to* t_k *and previous jobs arrive as late as legally allowed.*

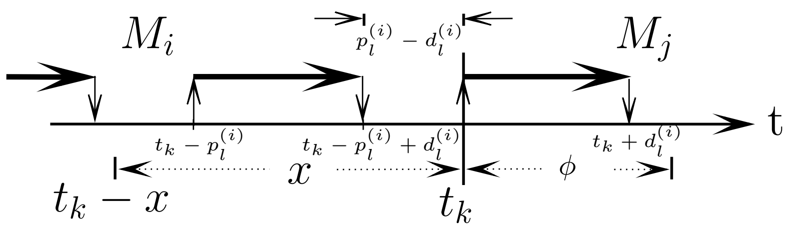

Figure 6.4: Jobs ($d_l^{(i)} \leq \phi$) that maximizes mcdbf..

The detailed proof for this lemma is identical to that of Lemma 4; therefore, we provide only a sketch as shown in figure 6.5. Assume a job sequence J' maximizes mode change demand which is not same as the one (let say J) described in Lemma 5. Figure 6.5 shows how J' can be transformed to J (as in Lemma 4) to prove this Lemma using induction.

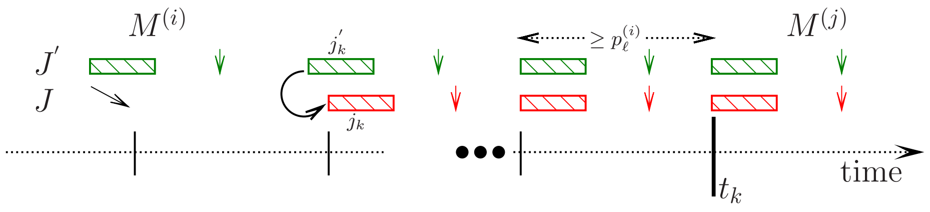

Figure 6.5: Worst case arrival for finished tasks ($d_\ell^{(i)} \leq \phi$). Downward arrow denotes deadline for each job, while rectangular region depicts execution requirements.

Lemmas 4 and 5 permit the calculation of the mode change carry-in demand for the non-aborted jobs using the following corollary. The corollary follows by simply counting the number of jobs in the sequences described by Lemmas 4 and 5.

Corollary 4 *For* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$, $\tau_\ell^{(i)} \in \tau^{(i)} \setminus \{\tau^{(ij)} \cup \alpha^{(ij)}\}$, *and* $x, \phi \geq 0$,

$$\mathsf{mcdbf}\left(\mathsf{mcr}_k, \tau_\ell^{(i)}, x, \phi\right) \leq \left\lfloor \frac{(x-\lambda_\ell^{(i)})_+}{p_\ell^{(i)}} \right\rfloor \cdot e_\ell^{(i)} + \mu_{\geq 0}\left(x - \lambda_\ell^{(i)}\right) \cdot e_\ell^{(i)} \tag{6.7}$$

where $\lambda_\ell^{(i)} \overset{\text{def}}{=} \left(d_\ell^{(i)} - \phi\right)_+$.

Proof: For the arrival sequence of Lemmas 4 and 5, observe that $t_k - \lambda_\ell^{(i)}$ corresponds to arrival of the last job of $\tau_\ell^{(i)}$ in $[t_k - x, t_k]$ that has deadline at or before $t_k + \phi$. If $x - \lambda_\ell^{(i)} < 0$, then no job of $\tau_\ell^{(i)}$ may arrive and have deadline in the interval $[t_k - x, t_k + \phi]$. Note that the right-hand-side of Equation 6.7 correctly evaluates to zero for this case.

If $x - \lambda_\ell^{(i)} \geq 0$, then we include the execution of the last job arriving at $t_k - \lambda_\ell^{(i)}$ (i.e., the second term of the right-hand-side of Equation 6.7) and the execution of jobs arriving and having deadline in the interval $[t_k - x, t_k - \lambda_\ell^{(i)}]$ (i.e., the first term of the right-hand-side of Equation 6.7) ■

We now describe the calculation of the demand for aborted jobs. The following lemma quantifies the highest demand from aborted jobs for a mode change request.

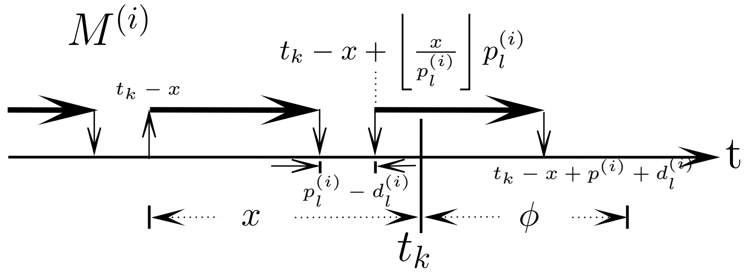
$M^{(i)}$
$t_k - x + \left\lfloor \frac{x}{p_l^{(i)}} \right\rfloor p_l^{(i)}$
$t_k - x$
t
$p_l^{(i)} - d_l^{(i)}$
$t_k - x + p^{(i)} + d_l^{(i)}$
x
ϕ
t_k

Figure 6.6: Aborted jobs maximizing mcdbf.

Lemma 6 *For* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$, $\tau_\ell^{(i)} \in \alpha^{(ij)}$, *and* $x, \phi \geq 0$,

$$\begin{aligned} &\mathsf{mcdbf}\left(\mathsf{mcr}_k, \tau_\ell^{(i)}, x, \phi\right) \\ &\leq \left\lfloor \frac{x}{p_\ell^{(i)}} \right\rfloor \cdot e_\ell^{(i)} + \mu_{\geq 0}\left((x - \left\lfloor \frac{x}{p_\ell^{(i)}} \right\rfloor p_\ell^{(i)}) + \phi - d_\ell^{(i)}\right) \\ &\quad \cdot \min\left(x - \left\lfloor \frac{x}{p_\ell^{(i)}} \right\rfloor p_\ell^{(i)}, e_\ell^{(i)}\right). \end{aligned} \tag{6.8}$$

Proof: The mcdbf function considers jobs that have both arrival and deadline in the given interval. A job from $\alpha^{(ij)}$ aborts immediately at the time of mode change request. For any interval of length x, at most $\left\lfloor \frac{x}{p_\ell^{(i)}} \right\rfloor$ jobs have their period completely contained in the x-length interval (i.e., if a job arrives at t and both t and $t+p_\ell^{(i)}$ are in the x-length interval, it is completely-contained). There are at most $\left\lceil \frac{x}{p_\ell^{(i)}} \right\rceil$ jobs that can arrive in such an interval (the last one may be only partially contained). Equation 6.8 includes the execution for the completely-contained jobs in the first term.

The last partially-contained job in the interval $[t_k - x, t_k]$ can arrive at the earliest at $t_k - x + \left\lfloor \frac{x}{p_\ell^{(i)}} \right\rfloor p_\ell^{(i)}$. If $x - \left\lfloor \frac{x}{p_\ell^{(i)}} \right\rfloor + \phi - d_\ell^{(i)}$ is positive, the last job ($\left\lceil \frac{x}{p_\ell^{(i)}} \right\rceil$-th) can get partial execution of at most $\min\left(x - \left\lfloor \frac{x}{p_\ell^{(i)}} \right\rfloor p_\ell^{(i)}, e_\ell^{(i)}\right)$ based on the interval length. Equation 6.8 accounts for $\left\lfloor \frac{x}{p_\ell^{(i)}} \right\rfloor$ complete jobs along with the last (partially-executed) job which proves our lemma. (A similar observation is made by Pedro and Burns [60] in the context of application-only mode changes.) Figure 6.6 illustrates the sequence described in the proof. ■

Now we consider the last set of tasks $\tau^{(ij)}$ which remains unaffected by the mode change request from $M^{(i)}$ to $M^{(j)}$. As there are no constraints on new job generation immediately after mode change, the mcdbf function for $\tau^{(ij)}$ represents the execution of the maximum number of jobs of $\tau^{(ij)}$ that can arrive and have deadline within the interval $[t_k - x, t_k + \phi]$. Note that this is the same as the dbf (according to Definition 3) and is summarized in the following lemma.

Lemma 7 *For* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$*,* $\tau_\ell^{(i)} \in \tau^{(ij)}$*, and* $x, \phi \geq 0$*,*

$$\mathsf{mcdbf}\left(\mathsf{mcr}_k, \tau_\ell^{(i)}, x, \phi\right) = \mathsf{dbf}(\tau_\ell^{(i)}, x + \phi). \tag{6.9}$$

6.1.4 Deriving the Carry-In Demand Function

We are now prepared to obtain an upper bound on the carry-in demand function, as described in the following lemma.

Lemma 8 *Consider* $\phi \geq 0$ *and successive mode change requests* $\mathsf{mcr}_0, \mathsf{mcr}_1, \ldots, \mathsf{mcr}_{k-1}, \mathsf{mcr}_k$ *where* $\mathsf{mcr}_{k-1} = \left(M^{(h)},\ M^{(i)},\ t_{k-1}\right)$ *and* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$*. If there are no deadline misses prior to* t_{k-1}*, we may obtain an upper-bound on the carry-in demand for mode-change request* mcr_k*,*

$$\mathsf{cidbf}(\mathsf{mcr}_k, \phi) \leq \sup_{0 \leq x \leq t_k - t_{k-1} - \delta^{(hi)}} \left\{ \begin{array}{l} \sum\limits_{\tau_\ell^{(i)} \in \tau^{(i)}} \mathsf{mcdbf}\left(\mathsf{mcr}_k, \tau_\ell^{(i)}, x, \phi\right) \\ + \mu_{\geq 0}\left(x - (t_k - t_{k-1} - \delta^{(hi)})\right) \cdot \mathsf{ci}(\mathsf{mcr}_{k-1}) \\ - \beta^i_{\mathsf{prior}}(x) \end{array} \right. \tag{6.10}$$

such that $\mathsf{ci}(\mathsf{mcr}_{k-1})$ *is upper bounded by the maximum of*

$$\sum_{\tau_\ell^{(hi)} \in \tau^{(hi)}} e_\ell^{(hi)} + \left(\begin{array}{l} \sum\limits_{\tau_\ell^{(hi)} \in \tau^{(hi)}} \left\lceil \frac{\left(\delta^{(hi)} - (p_\ell^{(hi)} - d_\ell^{(hi)})\right)}{p_\ell^{(hi)}} \right\rceil e_\ell^{(hi)} \\ + \sum\limits_{\tau_\ell^{(h)} \in \vartheta^{(hi)}} e_\ell^{(h)} - \beta^{h,i}_{\mathsf{trans}}(\delta^{(hi)}) \end{array} \right)_+ \tag{6.11}$$

and

$$\left(\begin{array}{l} \mathsf{cidbf}\left(\mathsf{mcr}_{k-1}, \max(\delta^{(hi)} + d^{(hi)}_{\max}, d^{(h)}_{\max})\right) \\ \quad - \sum_{\tau^{(hi)}_\ell \in \tau^{(hi)}} \mathsf{dbf}\left(\tau^{(hi)}_\ell, \max(\delta^{(hi)} + d^{(hi)}_{\max}, d^{(h)}_{\max}) - \delta^{(hi)} - p^{(hi)}_\ell\right) \\ -\beta^{h,i}_{\mathsf{trans}}(\delta^{(hi)}) \end{array}\right)_+ . \tag{6.12}$$

Proof:

Let t be the latest of time after $t_{k-1} + \delta^{(hi)}$ and the last time prior to t_k during which the processor is executing jobs with deadline later than $t_k + \phi$ (i.e., the processor is not busy executing jobs with deadline earlier than $t_k + \phi$). Let x be $t_k - t$. Thus, if t equals $t_{k-1} + \delta^{(hi)}$ (i.e., x equals $t_k - t_{k-1} - \delta^{(hi)}$), clearly, the amount of carry-in from $M^{(i)}$ to $M^{(j)}$ is at most the carry-in from mode $M^{(h)}$ (i.e., $\mathsf{ci}(\mathsf{mcr}_{k-1})$), plus the total demand generated minus the service received over $[t_{k-1} + \delta^{(hi)}, t_k + \phi]$. If t is later than $t_{k-1} + \delta^{(hi)}$ (i.e., $x < t_k - t_{k-1} - \delta^{(hi)}$), then the carry-in from $M^{(i)}$ to $M^{(j)}$ is at most the total demand generated minus the service received over $[t_{k-1} + \delta^{(hi)}, t_k + \phi]$. The upper bound for these two cases is quantified by Equation 6.10.

To see that Equation 6.11 is an upper bound on the carry-in from mode $M^{(h)}$ to $M^{(i)}$, observe that if there are no deadline misses prior to mcr_{k-1}, then all jobs with deadlines prior to t_{k-1} have completed execution. Thus, only (non-aborted) jobs of $\tau^{(h)}$ with deadline after t_{k-1} can contribute to the carry-in to $M^{(i)}$. (Note the constraint of Equation 5.2 prevents carry-in jobs from previous mode changes). For tasks of $\tau^{(hi)}$, the contribution of these tasks to $\mathsf{ci}(\mathsf{mcr}_{k-1})$ is maximized if each job of $\tau^{(hi)}_\ell \in \tau^{(hi)}$ arrives just prior to $t_{k-1} + \delta^{(hi)}$. This accounts for the first summation in Equation 6.11. For all tasks $\tau^{(h)}_\ell \in \vartheta^{(hi)}$, there is at most one job of $\tau^{(h)}_\ell$ active at time t_{k-1}. These jobs may contribute to the execution of $\mathsf{ci}(\mathsf{mcr}_{k-1})$ only if the total execution of $\vartheta^{(hi)}$ and jobs of $\tau^{(hi)}$ that may interfere over $[t_{k-1}, t_{k-1} + \delta^{(hi)})$ (given that the last job of $\tau^{(hi)}_\ell$ arrives just prior to $t_{k-1} + \delta^{(hi)}$) is greater than the total supply over the transition period. This accounts for the term inside the large $()_+$ in Equation 6.11.

To see that Equation 6.12 is also an upper bound on the carry-in from mode $M^{(h)}$ to $M^{(i)}$, observe that each job carried-in (according to Definition 14) from $M^{(h)}$ to $M^{(i)}$ must have a deadline by $t_{k-1} + \max(\delta^{(hi)} + d_{\max}^{(hi)}, d_{\max}^{(h)})$; therefore, $\mathsf{cidbf}\,(\mathsf{mcr}_{k-1},\ \max(\delta^{(hi)} + d_{\max}^{(hi)}, d_{\max}^{(h)}))$ is clearly an upper bound on the execution demand of jobs of $\tau^{(h)} \setminus \alpha^{(hi)}$ over the interval $[t_{k-1}, t_{k-1} + \delta^{(hi)}]$. Unfortunately, this expression may overestimate the contribution from unchanged tasks $\tau^{(hi)}$ since it could include jobs that arrive after $t_{k-1} + \delta^{(hi)}$. Therefore, we may need to subtract some execution from this expression. According to Equation 6.9 of Lemma 7, the execution contribution of an unchanged task $\tau_\ell^{(hi)}$ to $\mathsf{cidbf}(\mathsf{mcr}_{k-1}, \max(\delta^{(hi)} + d_{\max}^{(hi)}, d_{\max}^{(h)}))$ is upper-bounded by $\mathsf{mcdbf}(\mathsf{mcr}_{k-1}, \tau_\ell^{(hi)}, x, \max(\delta^{(hi)} + d_{\max}^{(hi)}, d_{\max}^{(h)}))$ for some $x > 0$ which is equal to $\mathsf{dbf}(\tau_\ell^{(hi)}, x + \max(\delta^{(hi)} + d_{\max}^{(hi)}, d_{\max}^{(h)}))$. Baruah et al. [8] show that the sequence that corresponds to this demand has a job of $\tau_\ell^{(hi)}$ arriving at $t_{k-1} - x$ and subsequent jobs as soon as legally possible. This sequence also corresponds to the maximum number of jobs that can arrive over $[t_{k-1} - x, t_{k-1} + \delta^{(hi)})$. In this sequence, the latest time that first job after $t_{k-1} + \delta^{(hi)}$ may arrive is $t_{k-1} + \delta^{(hi)} + p_\ell^{(hi)}$. Thus, subtracting the maximum execution of jobs of $\tau_\ell^{(hi)}$ arriving and having deadline in $[t_{k-1} + \delta^{(hi)} + p_\ell^{(hi)}, t_{k-1} + \max(\delta^{(hi)} + d_{\max}^{(hi)}, d_{\max}^{(h)})]$ gives us a more precise upper bound on the total execution for $\tau_\ell^{(hi)}$. Finally, we subtract the minimum execution received over $[t_{k-1}, t_{k-1} + \delta^{(hi)}]$ to obtain the bound expressed in Equation 6.12. ■

6.1.5 A Sufficient Schedulability Condition

For the schedulability analysis, we need to make sure the overall demand over any interval is always less than corresponding supply (i.e., $\beta_{\mathsf{post}}^{i,j}(x, y)$, $\beta_{\mathsf{trans}}^{i,j}(\phi)$, and $\beta_{\mathsf{prior}}^{i}(x)$). We will establish Theorem 2 which will check the schedulability conditions for a sequence of concrete mode change requests.

Theorem 2 *For a concrete sequence of mode-change requests* $\mathsf{mcr}_0, \mathsf{mcr}_1, \ldots$*, the subsystem is* EDF*-schedulable, if, for all* $k = 0, 1, \ldots$*, the following five conditions hold for* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$

$$\sum_{\tau_\ell^{(j)} \in \tau^{(j)}} \mathsf{dbf}(\tau_\ell^{(j)}, t) \le \mathsf{sbf}(\Omega^{(j)}, t),\ \forall t : 0 < t \le t_{k+1} - t_k - \delta^{(ij)}; \quad (6.13a)$$

$$\mathsf{cidbf}(\mathsf{mcr}_k, \phi) \le \beta^{i,j}_{\mathsf{trans}}(\phi),\ \forall \phi : 0 < \phi \le \delta^{(ij)}; \quad (6.13b)$$

$$\mathsf{cidbf}(\mathsf{mcr}_k, \delta^{(ij)} + t) + \sum_{\tau_\ell^{(j)} \in \tau^{(j)} \setminus \tau^{(ij)}} \mathsf{dbf}(\tau_\ell^{(j)}, t) - \beta^{i,j}_{\mathsf{trans}}(\delta^{(ij)}) \quad (6.13c)$$
$$\le \beta^{i,j}_{\mathsf{post}}(0, t),\ \forall t : 0 < t \le t_{k+1} - t_k - \delta^{(ij)};$$

$$\sum_{\tau_\ell^{(j)} \in \tau^{(j)} \setminus \tau^{(ij)}} \mathsf{dbf}(\tau_\ell^{(j)}, t) + \sum_{\tau_\ell^{(ij)} \in \tau^{(ij)}} \mathsf{dbf}(\tau_\ell^{(ij)}, t + s) \quad (6.13d)$$
$$\le \beta^{i,j}_{\mathsf{post}}(s, t),\ \forall s, t : 0 < t \le t_{k+1} - t_k - \delta^{(ij)}, 0 < s \le \delta^{(ij)};$$

$$\sum_{\tau_\ell^{(ij)} \in \tau^{(ij)}} \mathsf{dbf}(\tau_\ell^{(ij)}, t) \le \mathsf{sbf}(\Omega^{(ij)}, t),\ \forall t : 0 < t < \delta^{(ij)}; \quad (6.13e)$$

Proof: This theorem is proved by contrapositive. Assume the subsystem is not EDF-schedulable and misses its first deadline after the k-th mode change request $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$. This deadline miss can occur either during the transition period ($[t_k, t_k + \delta^{(ij)}]$) or during the operation

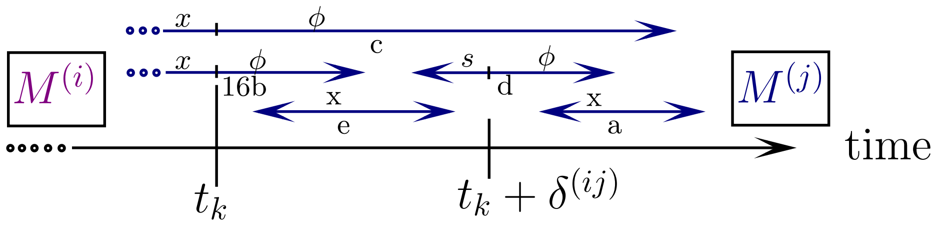

Figure 6.7: Arrows depict possible distributions of $[t_b, t_f]$

of $M^{(j)}$ after the transition. Suppose that $t_f < t_{k+1}$ denotes the earliest instant at which the EDF schedule misses the deadline. Without loss of generality, we further assume that t_b denotes the latest time-instant before t_f the subsystem is executing jobs with deadline greater than t_f or is idle (i.e., there are no backlogged jobs awaiting execution). During the interval $[t_b, t_f]$, the subsystem executes jobs arriving at or after t_b with deadline before or at t_f. These are precisely the jobs whose execution requirements contribute to demand over $[t_b, t_f]$. Let say $\eta(t_b, t_f)$ and $\beta(t_b, t_f)$ is denoting the maximum demand and the minimum supply (respectively) over the interval $[t_b, t_f]$. As a deadline miss occurs at t_f, it must be the case that $\eta(t_b, t_f) > \beta(t_b, t_f)$. We consider five different cases based t_f as follows (Figure 6.7):

1. $t_k + \delta^{(ij)} < t_f$**:** This case covers the scenario if deadline miss occurs during the normal operation of mode $M^{(j)}$. Depending on the position of t_b, we further consider the following two different cases:

   (a) $t_k + \delta^{(ij)} \leq t_b < t_f$**:** If t_b occurs after the mode change request and transition, tasks from $M^{(i)}$ are not allowed to generate new jobs. Therefore, only jobs from new-mode tasks contribute to $\eta(t_b, t_f)$. Furthermore, any carry-in from $M^{(i)}$ to $M^{(j)}$ with deadline earlier than t_f must have completed prior to t_b.

   $$\begin{aligned} &\eta(t_b, t_f) > \beta(t_b, t_f) \\ &\Rightarrow \mathsf{dbf}(\tau^{(j)}, t_f - t_b) > \mathsf{sbf}(\Omega^{(j)}, t_f - t_b) \\ &\Rightarrow \mathsf{dbf}(\tau^{(j)}, t) > \mathsf{sbf}(\Omega^{(j)}, t) \text{ where } t = t_f - t_b \end{aligned} \tag{6.13}$$

   The second line above follows from the fact that maximum execution requirements from $\tau^{(j)}$ over the interval $[t_b, t_f]$ is upper bounded by $\mathsf{dbf}(\tau^{(j)}, t_f - t_b)$ according to the definition of dbf. Similarly, $\mathsf{sbf}(\Omega^{(j)}, t)$ is a lower bound on the supply over $[t_b, t_f]$. The above equation is the negation of Equation 6.13a.

   (b) $t_b < t_k$**:** As t_b occurs during the operation of modes prior to $M^{(j)}$, the $\eta(t_b, t_f)$ may include jobs from all past modes and $M^{(j)}$; however, according to Equation 5.2, only the jobs from $M^{(i)}$ except aborted tasks can contribute to the carry-in at the mode

$M^{(j)}$ (otherwise, the first deadline could occur before t_f). Since t_b occurs earlier than t_k, the resource must be backlogged at time t_k. So, we may analyze the interval from t_k to t_f and replace each unfinished (non-aborted) job of $M^{(i)}$ at t_k by a new job with same absolute deadline and remaining execution requirements, but arrival time is redefined to t_k. These newly replaced jobs are the carry-in jobs; let J^c denotes the set of these carry-in jobs and all jobs generated from $\tau^{(ij)}$ in the interval $[t_k, t_k + \delta^{(ij)})$ with deadlines prior or at t_f. As there is a deadline miss at t_f, and the subsystem executes only jobs of $\tau^{(i)}$ and J^c, these jobs contribute to the demand over $[t_k, t_f]$. By assumption,

$$\begin{aligned}
&\eta(t_k, t_f) > \beta(t_k, t_f) \\
&\Rightarrow \mathsf{dbf}(\tau^{(j)} \setminus \tau^{(ij)}, t_f - t_k - \delta^{(ij)}) + \textstyle\sum_{j \in J^c} e_j \\
&\qquad > \beta^{i,j}_{\mathsf{trans}}(\delta^{(ij)}) + \beta^{i,j}_{\mathsf{post}}(0, t_f - t_k + \delta^{(ij)}).
\end{aligned}$$

An upper bound on carry-in demand and the demand from unchanged tasks $\tau^{(ij)}$ for the interval of length $t_k - t_f$ after mode change is $\mathsf{cidbf}(\mathsf{mcr}_k, t_f - t_k)$ according to the definition of cidbf. Furthermore, $\sum_{\tau_\ell^{(j)} \in \tau^{(j)} \setminus \tau^{(ij)}} \mathsf{dbf}(\tau_\ell^{(j)}, t)$ quantifies demand from the new tasks of $\tau^{(j)}$. If $t \overset{\text{def}}{=} t_f - t_k - \delta^{(ij)}$, then by the above discussion

$$\mathsf{cidbf}(\mathsf{mcr}_k, \delta^{(ij)} + t) - \beta^{i,j}_{\mathsf{trans}}(\delta^{(ij)}) + \sum_{\tau_\ell^{(j)} \in \tau^{(j)} \setminus \tau^{(ij)}} \mathsf{dbf}(\tau_\ell^{(j)}, t) > \beta^{i,j}_{\mathsf{post}}(0, t).$$

The above condition is the negation of 6.13c for $t \overset{\text{def}}{=} t_f - t_k - \delta^{(ij)}$.

(c) $t_k \le t_b < t_k + \delta^{(ij)}$**:** If t_b occurs at or after the mode change request, but before the transition is complete, we may observe that all carry-in jobs of $M^{(i)}$ with deadline prior to t_f have completed by t_b; otherwise, we would be in the previous case (i.e., $t_b < t_k$). Therefore, only jobs from new-mode tasks contribute to $\eta(t_b, t_f)$. By assumption,

$$\begin{aligned}
&\eta(t_b, t_f) > \beta(t_b, t_f) \\
&\Rightarrow \mathsf{dbf}(\tau^{(j)} \setminus \tau^{(ij)}, t_f - t_k - \delta^{(ij)}) + \mathsf{dbf}(\tau^{(ij)}, t_f - t_b) \\
&\qquad > \beta^{i,j}_{\mathsf{post}}(t_k + \delta^{(ij)} - t_b, t_f - t_k - \delta^{(ij)}) \\
&\Rightarrow \mathsf{dbf}(\tau^{(j)} \setminus \tau^{(ij)}, t) + \mathsf{dbf}(\tau^{(ij)}, t + s) > \beta^{i,j}_{\mathsf{post}}(s, t) \\
&\qquad\qquad \text{where } t = t_f - t_k - \delta^{(ij)} \text{ and } s = t_k + \delta^{(ij)} - t_b
\end{aligned} \tag{6.14}$$

The left-hand-side of the second line follows from the fact that jobs of $\tau^{(j)} \setminus \tau^{(ij)}$ cannot generate jobs until $t_k + \delta^{(ij)}$; thus, their maximum execution requirements over the interval $[t_b, t_f]$ is upper bounded by $\mathsf{dbf}(\tau^{(j)} \setminus \tau^{(ij)}, t_f - t_k - \delta^{(ij)})$ according to the definition of dbf. The above condition is the negation of 6.13d.

2. **$t_k < t_f \leq t_k + \delta^{(ij)}$:** This case covers the cases if deadline miss occurs during transition. For this there are two subcases depending on where t_b is located.

   (a) **$t_b \leq t_k$:** As t_b occurs during the operation of modes prior to $M^{(j)}$, this case is similar to the case (1)b; so, we analyze the interval from t_k to t_f. By assumption,

$$\begin{aligned}
&\eta(t_k, t_f) > \beta(t_k, t_f) \\
&\Rightarrow \mathsf{cidbf}(\mathsf{mcr}_k, t_f - t_k) > \beta^{i,j}_{\mathsf{trans}}(t_f - t_k) \\
&\Rightarrow \mathsf{cidbf}(\mathsf{mcr}_k, \phi) > \beta^{i,j}_{\mathsf{trans}}(\phi) \text{ where } \phi = t_f - t_k
\end{aligned}$$

   The above condition is the negation of 6.13b for $t \overset{\text{def}}{=} t_f - t_k - \delta^{(ij)}$.

   (b) **$t_k < t_b < t_f$:** If t_b occurs at or after the mode change request, we may observe that all carry-in jobs of $M^{(i)}$ with deadline prior to t_f have completed by t_b; otherwise, we would be in the previous case (i.e., $t_b < t_k$). Therefore, only unchanged jobs from new-mode tasks contribute to $\eta(t_b, t_f)$. The minimum supply for this interval is given by $\mathsf{sbf}(\Omega^{(ij)}, t_f - t_b)$.

$$\begin{aligned} &\eta(t_b, t_f) > \beta(t_b, t_f) \\ &\Rightarrow \mathsf{dbf}(\tau^{(ij)}, t_f - t_b) > \mathsf{sbf}(\Omega^{(ij)}, t_f - t_b) \\ &\Rightarrow \mathsf{dbf}(\tau^{(ij)}, t) > \mathsf{sbf}(\Omega^{(ij)}, t) \text{ where } t = t_f - t_b \end{aligned} \tag{6.15}$$

The above equation is the negation of Equation 6.13e.

With all of these cases, it is clear that the subsystem is EDF-schedulable if the subsystem satisfies above mentioned five conditions for all time intervals. ■

6.2 Non-Concrete Sequences of MCR

In the previous section, we obtained schedulability conditions for any concrete sequence of MCRs. In this section, we remove the assumption that the mode changes are a priori known and consider non-concrete sequences; the following summarizes our problem for this setting:

> EDF-Multi-Mode-Sched Problem: Given modes $M_1, \ldots, M^{(q)}$, resources $\Omega^{(ij)}$, offset $\delta^{(ij)}$, unchanged tasks $\tau^{(ij)}$, and aborted tasks $\alpha^{(ij)}$ for all $i, j \in \{1, \ldots, q\}$ $(i \neq j)$, determine whether all jobs (under all legal job arrival sequences and all possible legal mode-change requests according to Equations 5.1 and 5.2) are EDF-schedulable.

6.2.1 Definitions

The analysis of a non-concrete sequence will differ from the concrete sequence via the calculation of carry-in for past modes. For a concrete sequence, the maximum carry-in is determined from the fixed sequence of previous MCRs. As the exact time of an MCR is not known a priori for non-concrete sequences, the analysis needs to consider the maximum possible carry-in that could be generated from previous modes. That is, we must obtain an upper bound on any possible mode change from any mode $M^{(i)}$ to any other mode $M^{(j)}$ over all possible sequences of

MCRs, $\mathsf{mcr}_0, \mathsf{mcr}_1, \mathsf{mcr}_2, \ldots$. We now define equivalent carry-in execution and demand for a non-concrete sequence of MCRs.

Definition 11 (Non-Concrete Carry-In Execution) *The* non-concrete carry-in execution *from mode $M^{(i)}$ to any other mode $M^{(j)}$ at time t_k is an upper bound on the maximum possible remaining execution (over any legal sequence of MCRs) of non-aborted jobs from mode $M^{(i)}$ for tasks $\vartheta^{(ij)}$ (i.e., $\tau^{(i)} \setminus \{\tau^{(ij)} \cup \alpha^{(ij)}\}$) at time $t_k + \delta^{(ij)}$ that arrive prior to t_k and the maximum total execution of unchanged tasks (i.e., $\tau^{(ij)}$) that have arrival before $t_k + \delta^{(ij)}$. We denote this value by $\mathsf{ci}(M^{(i)}, M^{(j)})$ over any legal sequence of mode changes prior to t_k.*

Definition 12 (Non-Concrete Carry-In DBF) *The* non-concrete carry-in demand-bound function *for a mode change from $M^{(i)}$ to $M^{(j)}$ at time t_k and $\phi \in \mathbb{R}_{\geq 0}$ is the maximum remaining execution (over any legal sequence of MCRs prior to t_k) of jobs of tasks $\tau^{(i)} \setminus \alpha^{(ij)}$ that arrive prior to t_k (or prior to $t_k + \delta^{(ij)}$ for $\tau^{(ij)}$ tasks) and have deadline in the interval $[t_k, t_k + \phi]$. We denote this quantity by $\mathsf{cidbf}(M^{(i)}, M^{(j)}, \phi)$.*

6.2.2 A Sufficient Schedulability Test

Using a very similar concepts developed in the previous section, we now obtain demand supply relation for a non-concrete sequence of mode changes. The following lemma establishes an upper bound on the non-concrete carry-in demand.

Lemma 9 *For $\phi \geq 0$ and a mode change from $M^{(i)}$ to $M^{(j)}$, if all modes executing prior to the mode change did not miss a deadline, then*

$$\mathsf{cidbf}(M^{(i)}, M^{(j)}, \phi) \leq \widetilde{\Psi}\left(M^{(i)}, M^{(j)}, \phi, \mathsf{ci}(M^{(i)}, M^{(j)})\right) \tag{6.16}$$

where $\widetilde{\Psi}(M^{(i)}, M^{(j)}, \phi, \zeta)$ *is defined as*

$$\sup_{x>0}\left\{\begin{array}{l} \displaystyle\sum_{\tau_\ell^{(i)}\in\vartheta^{(ij)}} \left[\left\lfloor \frac{(x-\lambda_\ell^{(i)})_+}{p_\ell^{(i)}} \right\rfloor + 1\right] \times e_\ell^{(i)} \\ \displaystyle + \sum_{\tau_\ell^{(i)}\in\alpha^{(ij)}} \left\lfloor \frac{x}{p_\ell^{(i)}} \right\rfloor \times e_\ell^{(i)} + \sum_{\tau_\ell^{(i)}\in\alpha^{(ij)}} \min\left(x - \left\lfloor \frac{x}{p_\ell^{(i)}} \right\rfloor p_\ell^{(i)}, e_\ell^{(i)}\right) \\ \displaystyle + \sum_{\tau_\ell^{(i)}\in\tau^{(ij)}} \mathsf{dbf}(\tau_\ell^{(i)}, x+\phi) + \mu_{\geq 0}\left(x - N^{(i)}\Pi^{(i)}\right)\cdot\zeta - \beta^i_{\mathsf{prior}}(x) \end{array}\right. \tag{6.17}$$

and $\mathsf{ci}(M^{(i)}, M^{(j)})$ *is obtained from the convergence of the sequence* $\mathsf{ci}_0^{M^{(i)},M^{(j)}}$, $\mathsf{ci}_1^{M^{(i)},M^{(j)}}$, $\mathsf{ci}_2^{M^{(i)},M^{(j)}}$, *... for all* $i, j (i \neq j) \in \{1, \ldots, q\}$*. For any* $g \in \mathbb{N}$, $M^{(i)}$*, and* $M^{(j)}$ $(i \neq j)$*, if* $g = 0$*, then* $\mathsf{ci}_g^{M^{(i)},M^{(j)}} = 0$*; otherwise, if* $g > 0$*, then* $\mathsf{ci}_g^{M^{(i)},M^{(j)}}$ *is the minimum of* $W^{(ij)}$ *and* $f^{(ij)}(\max_{\substack{h=1,\ldots,q\\h\neq i}}\{\mathsf{ci}_{g-1}^{M^{(h)},M^{(i)}}\})$ *where* $W^{(ij)}$ *equals*

$$\sum_{\tau_\ell^{(ij)}\in\tau^{(ij)}} e_\ell^{(ij)} + \left(\sum_{\tau_\ell^{(ij)}\in\tau^{(ij)}} \left\lceil \frac{\left(\delta^{(ij)} - (p_\ell^{(ij)} - d_\ell^{(ij)})\right)}{p_\ell^{(ij)}} \right\rceil e_\ell^{(ij)} + \sum_{\tau_\ell^{(i)}\in\vartheta^{(ij)}} e_\ell^{(i)} - \beta^{i,j}_{\mathsf{trans}}(\delta^{(ij)}) \right)_+ \tag{6.18}$$

and $f^{(ij)}(\zeta)$ *is equal to*

$$\left(\begin{array}{l} \widetilde{\Psi}(M^{(i)}, M^{(j)}, \max(\delta^{(ij)} + d_{\max}^{(ij)}, d_{\max}^{(i)}), \zeta) \\ \displaystyle - \sum_{\tau_\ell^{(ij)}\in\tau^{(ij)}} \mathsf{dbf}\left(\tau_\ell^{(ij)}, \max(\delta^{(ij)} + d_{\max}^{(ij)}, d_{\max}^{(i)}) - \delta^{(ij)} - p_\ell^{(ij)}\right) - \beta^{i,j}_{\mathsf{trans}}(\delta^{(ij)}) \end{array}\right)_+ . \tag{6.19}$$

The convergence of the above sequence occurs at the smallest $g \in \mathbb{N}$ *such that* $\forall i, j (i \neq j) \in \{1, \ldots, q\}$, $\mathsf{ci}_g^{M^{(i)},M^{(j)}} = \mathsf{ci}_{g-1}^{M^{(i)},M^{(j)}}$.

Some remarks on Lemma 9: The function $f^{(ij)}(\zeta)$ calculates an upper bound on carry-in of a mode change request from $M^{(i)}$ (assuming the carry-in into $M^{(i)}$ from a previous mode is ζ) to the next mode $M^{(j)}$. This function acts in the very similar way to the Equation 6.12 for a concrete sequence of mode-change requests. Whereas Equation 6.12 evaluates only a finite number of x values, $f^{(ij)}(\zeta)$ invokes $\widetilde{\Psi}$ which evaluates all possible values of x as the potential MCR instants.

(This is necessary as the exact MCR instants are not known priori). Determining the carry-in after any sequence of mode-change requests for a non-concrete sequence is complex as the carry that each mode can forward to the next mode depends on the carry-in with which it starts with.

Before we prove Lemma 9, we need to prove some additional helper lemmas. In the next lemma, we show that if we have an upper bound on the carry-in for a previous mode change request, $\widetilde{\Psi}$ may be used as upper bound on cidbf for any mode change request.

Lemma 10 *For any sequence of mode changes,* $\mathsf{mcr}_0, \mathsf{mcr}_1, \ldots$*, consider* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$ *(where* $k \geq 1$*), if* $\zeta \geq \mathsf{ci}(\mathsf{mcr}_{k-1})$ *and* $\phi > 0$*, then* $\widetilde{\psi}(M^{(i)}, M^{(j)}, \phi, \zeta) \geq \mathsf{cidbf}(\mathsf{mcr}_k, \phi)$*.*

Proof: Observe that the right-hand-sides of cidbf and $\widetilde{\psi}$ within the $\sup$ expression are essentially equivalent functions except for the $\mu_{\geq 0}\left(x - (t_k - t_{k-1} - \delta^{(hi)})\right) \cdot \mathsf{ci}(\mathsf{mcr}_{k-1})$ in Equation 6.10 and the corresponding term $\mu_{\geq 0}\left(x - N_i \Pi^{(i)}\right) \cdot \zeta$ in Equation 6.17. However, the constraint of Equation 5.1 implies that $\mu_{\geq 0}\left(x - (t_k - t_{k-1} - \delta^{(hi)}\right)$ evaluates to one, only if $x \geq N_i \Pi^{(i)}$. Thus, if the μ of Equation 6.10 evaluates to one, then so does the μ of Equation 5.1. Since $\zeta \geq \mathsf{ci}(\mathsf{mcr}_{k-1})$, the function in the right-hand-side of Equation 6.17 is at least the value of the function on the right-hand-side of Equation 6.10 for the same value of x. Furthermore, the domain of x considered in the supremum of Equation 6.17 is a superset of the domain for Equation 6.10. Thus, clearly $\widetilde{\psi}(M^{(i)}, M^{(j)}, \phi, \zeta)$ is an upper bound on $\mathsf{cidbf}(\mathsf{mcr}_k, \phi)$. ■

The next lemma shows that $W^{(ij)}$ (i.e., Equation 6.18) is an upper bound on the amount of carry-in from mode $M^{(i)}$ to $M^{(j)}$.

Lemma 11 *For any sequence of mode changes* $\mathsf{mcr}_0, \mathsf{mcr}_1, \mathsf{mcr}_2, \ldots$*, consider a mode change from* $M^{(i)}$ *to* $M^{(j)}$ *(i.e.,* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$*). If the carry-in (*$\mathsf{ci}(\mathsf{mcr}_k)$*) from* $M^{(i)}$ *to* $M^{(j)}$ *is greater than* $W^{(ij)}$*, then some job generated in mode* $M^{(i)}$ *or during the transition (i.e., between* t_k *and* $t_k + \delta^{(ij)}$*) misses a deadline.*

Proof: We prove this lemma by contradiction. The proof is similar to the bound in Equation 6.11. Assume that the carry-in $\mathsf{ci}(\mathsf{mcr}_k)$ is greater than $W^{(ij)}$, but no job of $M^{(i)}$ misses a deadline. Only (non-aborted) jobs of $\tau^{(i)}$ with deadline after t_k can contribute to the carry-in to $M^{(i)}$, since

$d_\ell^{(i)} \leq p_\ell^{(i)}$ for all $\tau_\ell^{(i)} \in \tau^{(i)}$. (Note the constraint of Equation 5.2 prevents carry-in jobs from previous mode changes). For all tasks of $\tau^{(i)} \setminus \{\tau^{(ij)} \cup \alpha^{(ij)}\}$, there is at most one such job; the total execution of these jobs is $\sum_{\tau_\ell^{(i)} \in \tau^{(i)} \setminus \{\tau^{(ij)} \cup \alpha^{(ij)}\}} e_\ell^{(i)}$. For tasks $\tau_\ell^{(ij)}$ of $\tau^{(ij)}$, an upper bound on the execution that this task contributes to the carry-in is the execution requirement of $\tau_\ell^{(ij)}$ times the total number of jobs that can arrive in the interval $[t_k, t_k + \delta^{(ij)}]$ plus the execution of at most one job that can arrive prior to t_k. Adding together the total execution of all non-aborted jobs from $M^{(i)}$ and subtracting the minimum supply over the transition gives the upper bound of Equation 6.11. Thus, if more than $W^{(ij)}$ execution is carried-in, then it must come from a job generated prior to t_k with remaining execution. However, by the above discussion, this is not possible unless a job misses a deadline. ■

In the next two lemmas, we establish two properties of the function $\widetilde{\Psi}$ described in Equation 6.17.

Lemma 12 *For any $M^{(i)}$, $M^{(j)}$, and $\phi, \zeta \in \mathbb{N}$, the functions $\widetilde{\Psi}(M^{(i)}, M^{(j)}, \phi, \zeta)$, $W^{(ij)}$, and $f^{(ij)}(\zeta)$ are integer valued function.*

Proof: Lemma 12 always holds due to the fact that all task characteristics are natural integers. Furthermore, ceiling/floor function is used for calculating demand and supply. ■

Lemma 13 *For any $M^{(i)}$, $M^{(j)}$, and $\phi > 0$, the functions $\widetilde{\Psi}(M^{(i)}, M^{(j)}, \phi, \zeta)$ and $f^{(ij)}(\zeta)$ are monotonically non-decreasing on ζ.*

Proof: This lemma follows from the fact that ζ is directly added in the definition of $\widetilde{\Psi}(M^{(i)}, M^{(j)}, \phi, \zeta)$ depending on the value of $N^{(i)}$ and $\Pi^{(i)}$ which is independent of ζ; so, there is no way of getting lower value from the function $\widetilde{\Psi}(M^{(i)}, M^{(j)}, \phi, \zeta)$ for any pair of modes $M^{(i)}$ and $M^{(j)}$ at a given ϕ with higher value of ζ than that of smaller ζ. Since $f^{(ij)}(\zeta)$ changes only with $\widetilde{\Psi}$ (the remainder is fixed), $f^{(ij)}$ is also monotonically non-decreasing. ■

For any pair of modes $M^{(i)}$ and $M^{(j)}$, the Equations 6.18 and 6.19 calculate the maximum carry-in with which $M^{(i)}$ can start with, and can affect the demand even after mode change to the next mode $M^{(j)}$. Using these two equations, we show that the carry-in is always bounded from the above by $\mathsf{ci}(M^{(i)}, M^{(j)})$.

Lemma 14 *For any sequence of ℓ mode changes, $\mathsf{mcr}_0, \mathsf{mcr}_1, \ldots, \mathsf{mcr}_\ell$, the carry-in after the $\mathsf{mcr}_\ell = (M^{(i)}, M^{(j)}, t_\ell)$ must be less or equal to $\mathsf{ci}_\ell^{M^{(i)},M^{(j)}}$ if the subsystem does not miss any deadline before t_ℓ.*

Proof: The proof is by induction on ℓ.

Base Case: The base case is $\ell = 1$. To show that $\mathsf{ci}(\mathsf{mcr}_1) \leq \mathsf{ci}_1^{M^{(i)},M^{(j)}}$, we must show that both $\mathsf{ci}(\mathsf{mcr}_1) \leq W^{(ij)}$ and $\mathsf{ci}(\mathsf{mcr}_1) \leq f^{(ij)}(0)$ are satisfied. Note that $\mathsf{ci}(\mathsf{mcr}_0)$ is equal to zero. By the fact that there are no deadline misses prior to t_1 and Lemma 11, the first condition is satisfied. For the second condition, Lemma 10 implies that $\widetilde{\psi}(M^{(i)}, M^{(j)}, \phi, 0) \geq \mathsf{cidbf}(\mathsf{mcr}_1, \phi)$ for all $\phi \geq 0$ since $0 \geq \mathsf{ci}(\mathsf{mcr}_0)$. As the right-hand-side of Equation 6.12 for $\mathsf{ci}(\mathsf{mcr}_1)$ is identical to $f^{(ij)}(0)$ except for the first term (i.e., $\mathsf{cidbf}(\mathsf{mcr}_1, \max(\delta^{(ij)} + d_{\max}^{(ij)}, d_{\max}^{(i)}))$ versus $\widetilde{\Psi}(M^{(i)}, M^{(j)}, \max(\delta^{(ij)} + d_{\max}^{(ij)}, d_{\max}^{(i)}), 0))$, $f^{(ij)}(0)$ is clearly an upper bound on $\mathsf{ci}(\mathsf{mcr}_1)$. The base case follows.

Induction hypothesis: Assume that the carry-in is always less than $\mathsf{ci}_\ell^{M^{(i)},M^{(j)}}$ for any sequence of length ℓ or less non concrete mode change requests.

Induction: The $\mathsf{ci}_{\ell+1}^{M^{(i)},M^{(j)}}$ is the minimum of $W^{(ij)}$ and $f^{(ij)}(\mathsf{ci}_\ell^{M^{(i)},M^{(j)}})$. In case $\mathsf{ci}_{\ell+1}^{M^{(i)},M^{(j)}}$ is determined by $W^{(ij)}$, the induction step follows trivially as $\mathsf{ci}(\mathsf{mcr}_{\ell+1})$ must be always less or equal to $W^{(ij)}$ for a schedulable subsystem (by Lemma 11). Thus, we must consider if $\mathsf{ci}_\ell^{M^{(i)},M^{(j)}}$ corresponds to $f^{(ij)}(\max_{h=1,\ldots,q \wedge h \neq i} \{\mathsf{ci}_\ell^{M^{(i)},M^{(j)}}\})$. The function $f^{(ij)}(\zeta)$ is monotonically non-decreasing on ζ. By induction hypothesis $\mathsf{ci}(\mathsf{mcr}_\ell) \leq \mathsf{ci}_\ell^{M^{(i)},M^{(j)}}$. Thus, $\max_{h=1,\ldots,q \wedge h \neq i} \{\mathsf{ci}_\ell^{M^{(h)},M^{(i)}}\} \geq \mathsf{ci}(\mathsf{mcr}_\ell)$. Thus, Lemma 10 implies that $\widetilde{\psi}(M^{(i)}, M^{(j)}, \phi, \max_{h=1,\ldots,q \wedge h \neq i} \{\mathsf{ci}_\ell^{M^{(h)},M^{(i)}}\}) \geq \mathsf{cidbf}(\mathsf{mcr}_\ell, \phi)$ for all $\phi \geq 0$. As the right-hand-side of Equation 6.12 for $\mathsf{ci}(\mathsf{mcr}_\ell)$ is identical to $f^{(ij)}(\max_{h=1,\ldots,q \wedge h \neq i} \{\mathsf{ci}_\ell^{M^{(h)},M^{(i)}}\})$ except for the first term, $f^{(ij)}(\max_{h=1,\ldots,q \wedge h \neq i} \{\mathsf{ci}_\ell^{M^{(h)},M^{(i)}}\}) \geq \mathsf{ci}_\ell^{M^{(i)},M^{(j)}}$ implies that $\mathsf{ci}_\ell^{M^{(i)},M^{(j)}}$ is clearly an upper bound on $\mathsf{ci}(\mathsf{mcr}_1)$. The theorem follows.

■

Corollary 5 *For any pair of modes $M^{(i)}$ and $M^{(j)}$, the convergence $\mathsf{ci}^{M^{(i)},M^{(j)}}$ of the sequence $\mathsf{ci}_0^{M^{(i)},M^{(j)})}$, $\mathsf{ci}_1^{M^{(i)},M^{(j)}}$, $\mathsf{ci}_2^{M^{(i)},M^{(j)}}$, ... exists.*

Proof: The corollary follows from the fact that $\mathsf{ci}_\ell^{M^{(i)},M^{(j)}}$ is both integer-valued and monotonically non-decreasing in ℓ, as $W^{(ij)}$ and $f^{(ij)}$ are integer-valued and monotonically non-decreasing (by Lemmas 12 and 13). Furthermore, $\mathsf{ci}_\ell^{M^{(i)},M^{(j)}}$ is upper bounded by $W^{(ij)}$ implying that it must converge.

■

Corollary 6 *For any sequence of mode changes* $\mathsf{mcr}_0, \mathsf{mcr}_1, \ldots$*, consider* $\mathsf{mcr}_\ell = (M^{(i)}, M^{(j)}, t_\ell)$*, it must be that* $\mathsf{ci}^{M^{(i)},M^{(j)}} \geq \mathsf{ci}(\mathsf{mcr}_\ell)$ *for all* $\ell \in \mathbb{N}$.

Proof: By Lemma 14, $\mathsf{ci}_\ell^{M^{(i)},M^{(j)}} \geq \mathsf{ci}(\mathsf{mcr}_\ell)$. Corollary shows the sequence of ci_ℓ is monotonically non-decreasing and converges. Thus, $\mathsf{ci}^{M^{(i)},M^{(j)}} \geq \mathsf{ci}_\ell^{M^{(i)},M^{(j)}}$ implying the corollary.

■

We are finally prepared to prove Lemma 9.

Proof of Lemma 9 Consider any mode change $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$. Corollary 6 implies that $\mathsf{ci}^{M^{(i)},M^{(j)}} \geq \mathsf{ci}(\mathsf{mcr}_k)$. By Lemma 10, $\widetilde{\psi}(M^{(i)}, M^{(j)}, \phi, \mathsf{ci}^{M^{(i)},M^{(j)}}) \geq \mathsf{cidbf}(\mathsf{mcr}_k, \phi)$. Since $\mathsf{cidbf}(M^{(i)}, M^{(j)}, \phi)$ equals the carry-in possible over all such mcr_k, it must be that $\widetilde{\psi}(M^{(i)}, M^{(j)}, \phi, \mathsf{ci}^{M^{(i)},M^{(j)}}) \geq \mathsf{cidbf}(M^{(i)}, M^{(j)}, \phi)$ is also true.

■

The following theorem on schedulability for non-concrete MCRs follows from repeated applications of Lemma 9, the observation that $\widetilde{\Psi}$ is monotonically non-decreasing, and Theorem 2. The proof is very similar to the Theorem 2; However, the detailed proof is included in the following for making the thesis complete.

Theorem 3 *For any possible sequence of mode-change requests, the subsystem is* EDF*-schedulable, if the following five conditions hold for any two distinct modes $M^{(i)}$ and $M^{(j)}$,*

$$\sum_{\tau_\ell^{(j)} \in \tau^{(j)}} \mathsf{dbf}(\tau_\ell^{(j)}, t) \le \mathsf{sbf}(\Omega^{(j)}, t),\ \forall t > 0, \tag{6.20a}$$

$$\widetilde{\Psi}(M^{(i)}, M^{(j)}, \phi, \mathsf{ci}^{M^{(i)}, M^{(j)}}) \le \beta^{i,j}_{\mathsf{trans}}(\phi)$$
$$\forall \phi : 0 < \phi \le \delta^{(ij)}, \tag{6.20b}$$

$$\widetilde{\Psi}(M^{(i)}, M^{(j)}, \delta^{(ij)} + t, \mathsf{ci}^{M^{(i)}, M^{(j)}}) - \beta^{i,j}_{\mathsf{trans}}(\delta^{(ij)}) \tag{6.20c}$$
$$+ \sum_{\tau_\ell^{(j)} \in \tau^{(j)} \setminus \tau^{(ij)}} \mathsf{dbf}(\tau_\ell^{(j)}, t) \le \beta^{i,j}_{\mathsf{post}}(0, t),\ \forall t > 0$$

$$\sum_{\tau_\ell^{(j)} \in \tau^{(j)} \setminus \tau^{(ij)}} \mathsf{dbf}(\tau_\ell^{(j)}, t) + \sum_{\tau_\ell^{(ij)} \in \tau^{(ij)}} \mathsf{dbf}(\tau_\ell^{(ij)}, t + s) \tag{6.20d}$$
$$\le \beta^{i,j}_{\mathsf{post}}(s, t),\ \forall s, t : 0 < t, 0 < s \le \delta^{(ij)}$$

$$\sum_{\tau_\ell^{(ij)} \in \tau^{(ij)}} \mathsf{dbf}(\tau_\ell^{(ij)}, t) \le \mathsf{sbf}(\Omega_{(ij)}, t),\ \forall t : 0 < t < \delta^{(ij)}, \tag{6.20e}$$

Proof: This theorem is proved by contrapositive. The minor difference with Theorem 2 is that the sequence of mode changes are not fixed. Assume the subsystem is not EDF-schedulable and misses its first deadline while executing mode $M^{(j)}$ after the k-th mode change request at time t_k (i.e., $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$). Let $\mathsf{mcr}_0, \mathsf{mcr}_1, \ldots, \mathsf{mcr}_{k-1}, \mathsf{mcr}_k$ be the sequence of MCRs that led to the deadline miss. This deadline miss can occur either during the transition period ($[t_k, t_k + \delta^{(ij)}]$) or during the operation of $M^{(j)}$ after the transition. Suppose that $t_f < t_{k+1}$ denotes the earliest instant at which the EDF schedule misses the deadline. Without loss of generality, we further assume that t_b is denoting the latest time-instant before t_f the subsystem is executing jobs with deadline greater than t_f or is idle (i.e., there are no backlogged jobs awaiting execution). During the interval $[t_b, t_f]$, the subsystem executes jobs arriving at or after t_b with deadline before or at t_f. These are precisely the jobs whose execution requirements contribute to demand over $[t_b, t_f]$. Let say $\eta(t_b, t_f)$ and $\beta(t_b, t_f)$ is denoting the maximum demand and the minimum supply over the interval $[t_b, t_f]$. As a deadline miss occurs at t_f, it must be the case that $\eta(t_b, t_f) > \beta(t_b, t_f)$.

In the following section, we consider different cases based on the location of t_f.

1. $t_k + \delta^{(ij)} < t_f$**:**

   This case covers the scenario if a deadline miss occurs during the normal operation of a mode $M^{(j)}$. Depending on the position of t_b, we further consider the following two different cases:

   (a) $t_k + \delta^{(ij)} \leq t_b < t_f$**:**

   If t_b occurs after the mode change request and transition, tasks from $M^{(i)}$ are not allowed to generate new jobs. Therefore, only jobs from new-mode tasks contribute to $\eta(t_b, t_f)$. Furthermore, any more carry-in from $M^{(i)}$ to $M^{(j)}$ with deadline earlier than t_f must have completed prior to t_b.

   $$\begin{aligned} &\eta(t_b, t_f) > \beta(t_b, t_f) \\ &\Rightarrow \mathsf{dbf}(\tau^{(j)}, t_f - t_b) > \mathsf{sbf}(\Omega^{(j)}, t_f - t_b) \\ &\Rightarrow \mathsf{dbf}(\tau^{(j)}, t) > \mathsf{sbf}(\Omega^{(j)}, t) \text{ where } t = t_f - t_b \end{aligned} \tag{6.20}$$

   The second line above follows from the fact that maximum execution requirements from $\tau^{(j)}$ over the interval $[t_b, t_f]$ is upper bounded by $\mathsf{dbf}(\tau^{(j)}, t_f - t_b)$ according to the definition of dbf. Similarly, $\mathsf{sbf}(\Omega^{(j)}, t)$ is a lower bound on the supply over $[t_b, t_f]$. The above equation is the negation of Equation 6.13a.

   (b) $t_b < t_k$**:**

   As t_b occurs during the operation of the previous mode $M^{(i)}$, $\eta(t_b, t_f)$ may include jobs from both $M^{(i)}$ and $M^{(j)}$. Except for unchanged tasks $\tau^{(ij)}$, any other task from $M^{(i)}$ cannot generate new jobs after t_k. So, we may analyze the interval from t_k to t_f and replace each unfinished (non-aborted) job of $M^{(i)}$ at t_k by a new job with same absolute deadline and remaining execution requirements, but the arrival time is redefined to t_k. These newly replaced jobs are the carry-in jobs; let J^c denotes the set of these carry-in jobs and all jobs generated from $\tau^{(ij)}$ in the interval $[t_k, t_k + \delta^{(ij)})$ with deadlines prior or at t_f. As there is a deadline miss at t_f, and the subsystem executes only jobs of $\tau^{(i)}$ and J^c, these jobs contribute to the demand over $[t_k, t_f]$. By assumption,

$$\begin{aligned}
&\eta(t_k, t_f) > \beta(t_k, t_f) \\
&\Rightarrow \mathsf{dbf}(\tau^{(j)} \setminus \tau^{(ij)}, t_f - t_k - \delta^{(ij)}) + \textstyle\sum_{j \in J^c} e_j \\
&\quad > \beta^{i,j}_{\mathsf{trans}}(\delta^{(ij)}) + \beta^{i,j}_{\mathsf{post}}(0, t_f - t_k + \delta^{(ij)}).
\end{aligned}$$

An upper bound on carry-in demand and the demand from unchanged tasks $\tau^{(ij)}$ for the interval of length $t_k - t_f$ after mode change is $\mathsf{cidbf}(\mathsf{mcr}_k, t_f - t_k)$ according to the definition of cidbf. Furthermore, $\widetilde{\psi}(M^{(i)}, M^{(j)}, t_f - t_k, \mathsf{ci}^{M^{(i)},M^{(j)}})$ due to Lemma 9. $\sum_{\tau_\ell^{(j)} \in \tau^{(j)} \setminus \tau^{(ij)}} \mathsf{dbf}(\tau_\ell^{(j)}, t)$ quantifies demand from the new tasks of $\tau^{(j)}$. If $t \stackrel{\text{def}}{=} t_f - t_k - \delta^{(ij)}$, then by the above discussion

$$\begin{aligned}
&\widetilde{\psi}(M^{(i)}, M^{(j)}, t + \delta^{(ij)}, \mathsf{ci}^{M^{(i)},M^{(j)}}) - \beta^{i,j}_{\mathsf{trans}}(\delta^{(ij)}) \\
&\qquad + \sum_{\tau_\ell^{(j)} \in \tau^{(j)} \setminus \tau^{(ij)}} \mathsf{dbf}(\tau_\ell^{(j)}, t) > \beta^{i,j}_{\mathsf{post}})(0, t).
\end{aligned}$$

The above condition is the negation of 6.13c for $t \stackrel{\text{def}}{=} t_f - t_k - \delta^{(ij)}$.

(c) **$t_k \le t_b < t_k + \delta^{(ij)}$:**

If t_b occurs at or after the mode change request, but before the transition is complete, we may observe that all carry-in jobs of $M^{(i)}$ with deadline prior to t_f have completed by t_b; otherwise, we would be in the previous case (i.e., $t_b < t_k$). Therefore, only jobs from new-mode tasks contribute to $\eta(t_b, t_f)$. By assumption,

$$\begin{aligned}
&\eta(t_b, t_f) > \beta(t_b, t_f)\\
&\Rightarrow \mathsf{dbf}(\tau^{(j)} \setminus \tau^{(ij)}, t_f - t_k - \delta^{(ij)}) + \mathsf{dbf}(\tau^{(ij)}, t_f - t_b)\\
&\qquad > \beta^{i,j}_{\mathsf{post}}(t_k + \delta^{(ij)} - t_b, t_f - t_k - \delta^{(ij)})\\
&\Rightarrow \mathsf{dbf}(\tau^{(j)} \setminus \tau^{(ij)}, t) + \mathsf{dbf}(\tau^{(ij)}, t + s)\\
&\qquad > \beta^{i,j}_{\mathsf{post}}(s, t) \text{ where } t = t_f - t_k - \delta^{(ij)}\\
&\qquad\qquad \text{and } s = t_k + \delta^{(ij)} - t_b
\end{aligned} \tag{6.21}$$

The left-hand-side of the second line follows from the fact that jobs of $\tau^{(j)} \setminus \tau^{(ij)}$ cannot generate jobs until $t_k + \delta^{(ij)}$; thus, their maximum execution requirements over the interval $[t_b, t_f]$ is upper bounded by $\mathsf{dbf}(\tau^{(j)} \setminus \tau^{(ij)}, t_f - t_k - \delta^{(ij)})$ according to the definition of dbf. The above condition is the negation of 6.13d.

2. $t_k < t_f \leq t_k + \delta^{(ij)}$**:**

   This case covers the cases if deadline miss occurs during transition. For this there are two subcases depending on where t_b is located.

   (a) $t_b \leq t_k$**:**

   As t_b occurs during the operation of the previous mode $M^{(i)}$, $\eta(t_b, t_f)$ may include only non-aborted and unchanged jobs from $M^{(i)}$. In other words, the demand over $[t_b, t_f]$ equals the carry-in and unchanged jobs of $M^{(i)}$ with deadline at or before t_f.

$$\begin{aligned}
&\eta(t_b, t_f) > \beta(t_b, t_f)\\
&\Rightarrow \mathsf{cidbf}(\mathsf{mcr}_k, t_f - t_k) > \beta^{i,j}_{\mathsf{trans}}(t_f - t_k)\\
&\Rightarrow \mathsf{cidbf}(\mathsf{mcr}_k, \phi) > \beta^{i,j}_{\mathsf{trans}}(\phi) \text{ where } \phi = t_f - t_k\\
&\Rightarrow \widetilde{\psi}(M^{(i)}, M^{(j)}, \phi, \mathsf{ci}^{M^{(i)}, M^{(j)}}) > \beta^{i,j}_{\mathsf{trans}}(\phi)
\end{aligned}$$

   The above condition is the negation of 6.13b for $t \overset{\text{def}}{=} t_f - t_k - \delta^{(ij)}$.

   (b) $t_k < t_b < t_f$**:**

   If t_b occurs at or after the mode change request, we may observe that all carry-in jobs of $M^{(i)}$ with deadline prior to t_f have completed by t_b; otherwise, we would be in the previous case (i.e., $t_b < t_k$). Therefore, only unchanged jobs from new-mode tasks contribute to $\eta(t_b, t_f)$. The minimum supply for this interval is given by $\mathsf{sbf}(\Omega^{(ij)}, t_f - t_b)$.

$$\begin{aligned}
&\eta(t_b, t_f) > \beta(t_b, t_f) \\
&\Rightarrow \mathsf{dbf}(\tau^{(ij)}, t_f - t_b) > \mathsf{sbf}(\Omega^{(ij)}, t_f - t_b) \\
&\Rightarrow \mathsf{dbf}(\tau^{(ij)}, t) > \mathsf{sbf}(\Omega^{(ij)}, t) \text{ where } t = t_f - t_b
\end{aligned} \tag{6.22}$$

The above equation is the negation of Equation 6.13e.

With all of these cases, it is clear that the subsystem is EDF-schedulable if the subsystem satisfies above mentioned five conditions for all time intervals. ■

6.2.3 Reducing the Time Complexity

For finding the worst-case mode-change carry-in demand, Theorem 3, as written, has to evaluate potentially unbounded number of values of t for Equation 6.20. Furthermore, it is also not immediately obvious how to efficiently compute $\widetilde{\Psi}$ from Lemma 9 as it requires evaluating an expression over any infinite number of values for x and iteratively computing a converging sequence of values for ci. In this subsection, we derive more efficient time bounds for our schedulability test. The next section will use the lemmas derived in this section to efficiently implement our schedulability test for non-concrete MCRs.

In the following three lemmas, we obtain upper bounds on times for which the right-hand-side of Equation 6.17 need to be evaluated. The results are inspired by similar bounds obtained by Baruah et al. [9] for uni-modal systems. We will abuse notation below and assume that a zero in the denominator of a fraction evaluates to ∞. We will also assume that for each $M^{(i)}$ the utilization $u^{(i)}$ is at most $\Theta^{(i)}/\Pi^{(i)}$ as this is a necessary condition for schedulability on a periodic resource [70].

Lemma 15 *For $\phi \geq 0$, any $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$ in arbitrary sequence of mode-change requests, and $p^{(i)}_{max} \overset{def}{=} \max_{\tau^{(i)}_\ell \in \tau^{(i)}}\{p^{(i)}_\ell\}$, if $\widetilde{\Psi}(M^{(i)}, M^{(j)}, \phi, \zeta)$ (Equation 6.17) is at least $\xi \geq 0$, then the x that maximizes the supremum in the right-hand-side of Equation 6.17 occurs at or before the maximum of $d^{(i)}_{\max}$ and the minimum of $H^{(i)} + d^{(i)}_{\max} + N^{(i)}\Pi^{(i)}$ and*

$$\left\lceil \frac{u^{(i)} \cdot p^{(i)}_{max} + u^{(ij)}\phi + \zeta + \frac{\Theta^{(i)}(\Pi^{(i)} - \Theta^{(i)})}{\Pi^{(i)}} - \xi}{\frac{\Theta^{(i)}}{\Pi^{(i)}} - u^{(i)}} \right\rceil . \tag{6.23}$$

where $H^{(i)} \overset{def}{=} \mathsf{lcm}\left\{\{p^{(i)}_\ell\}_{\tau^{(i)}_\ell \in \tau^{(i)}} \cup \{\Pi^{(i)}\}\right\}$.

Proof: We consider two cases: 1) $u^{(i)}$ equals $\Theta^{(i)}/\Pi^{(i)}$; and 2) $u^{(i)}$ is strictly less than $\Theta^{(i)}/\Pi^{(i)}$. In the first case, we use techniques similar to [9] to show that $H^{(i)} + d^{(i)}_{\max} + N^{(i)}\Pi^{(i)}$ is an upper bound on x. Let $\widetilde{\Psi}_x$ be the expression inside the $\sup$ of Equation 6.17 for a given $x > 0$. Assume that $\widetilde{\Psi}_x$ obtains its maximum value at some x equal to $t + a \cdot H^{(i)} + d^{(i)}_{\max} + N^{(i)}\Pi^{(i)}$ where $a \in \mathbb{Z}^+$ and $0 \leq t < H^{(i)}$. Let x' equal $t + d^{(i)}_{\max} + N^{(i)}\Pi^{(i)}$. By definition, $\widetilde{\Psi}_x$ equals

$$\begin{aligned}
&\sum_{\tau^{(i)}_\ell \in \vartheta^{(ij)}} \left[\left\lfloor \frac{(x' + H^{(i)} - \lambda^{(i)}_\ell)_+}{p^{(i)}_\ell} \right\rfloor + 1\right] \times e^{(i)}_\ell + \sum_{\tau^{(i)}_\ell \in \alpha^{(ij)}} \left\lfloor \frac{x' + H^{(i)}}{p^{(i)}_\ell} \right\rfloor \times e^{(i)}_\ell \\
&+ \sum_{\tau^{(i)}_\ell \in \alpha^{(ij)}} \min\left(x' + H^{(i)} - \left\lfloor \frac{x' + H^{(i)}}{p^{(i)}_\ell} \right\rfloor p^{(i)}_\ell, e^{(i)}_\ell\right) \\
&+ \sum_{\tau^{(i)}_\ell \in \tau^{(ij)}} \mathsf{dbf}(\tau^{(i)}_\ell, x' + H^{(i)} + \phi) + \zeta - \ \beta^{(}_{\mathsf{prior}} M^{(i)}, x' + H^{(i)}).
\end{aligned}$$

Since $H^{(i)}$ divided by any $p^{(i)}_\ell$ or $\Pi^{(i)}$ is an integer, we have the following that $\widetilde{\Psi}_{x'} + H^{(i)} \sum_{\tau^{(i)}_\ell \in \tau^{(i)}} \frac{e^{(i)}_\ell}{p^{(i)}_\ell} - H^{(i)} \frac{\Theta^{(i)}}{\Pi^{(i)}}$. However, we have assumed that $u^{(i)}$ equals $\Theta^{(i)}/\Pi^{(i)}$; this implies that $\widetilde{\Psi}_{x'}$ equals $\widetilde{\Psi}_x$. Thus, we have shown in this case that the maximum occurs prior to $H^{(i)} + d^{(i)}_{\max} + N^{(i)}\Pi^{(i)}$.

For the second case, when $u^{(i)}$ is strictly less than $\Theta^{(i)}/\Pi^{(i)}$ we may obtain a potentially tighter upper bound. Suppose that the right-hand side of Equation 6.17 obtains its supremum at some $x > d^{(i)}_{\max}$. According to Equation 6.17, we have

$$
\begin{aligned}
\xi &< \sum_{\tau_\ell^{(i)} \in \vartheta^{(ij)}} \left[\left\lfloor \frac{(x-\lambda_\ell^{(i)})_+}{p_\ell^{(i)}} \right\rfloor + 1 \right] \times e_\ell^{(i)} + \zeta + \sum_{\tau_\ell^{(i)} \in \tau^{(ij)}} \mathsf{dbf}(\tau_\ell^{(i)}, x+\phi) \\
&\quad + \sum_{\tau_\ell^{(i)} \in \alpha^{(ij)}} \left\lfloor \frac{x}{p_\ell^{(i)}} \right\rfloor \times e_\ell^{(i)} + \sum_{\tau_\ell^{(i)} \in \alpha^{(ij)}} \min\left(x - \left\lfloor \frac{x}{p_\ell^{(i)}} \right\rfloor p_\ell^{(i)}, e_\ell^{(i)} \right) - \beta^i_{\mathsf{prior}}(x)
\end{aligned}
$$

$$
\begin{aligned}
\Rightarrow \xi &< \sum_{\tau_\ell^{(i)} \in \vartheta^{(ij)}} \left[\frac{(x-\lambda_\ell^{(i)})_+}{p_\ell^{(i)}} + 1 \right] \times e_\ell^{(i)} + \zeta - \beta^i_{\mathsf{prior}}(x) \\
&\quad + \sum_{\tau_\ell^{(i)} \in \tau^{(ij)}} \left(\frac{x+\phi-d_\ell^{(i)}}{p_\ell^{(i)}} + 1 \right)_+ e_\ell^{(i)} + \sum_{\tau_\ell^{(i)} \in \alpha^{(ij)}} \frac{x}{p_\ell^{(i)}} \times e_\ell^{(i)} + \sum_{\tau_\ell^{(i)} \in \alpha^{(ij)}} e_\ell^{(i)}
\end{aligned}
$$

$$
\begin{aligned}
\Rightarrow \xi &< \sum_{\tau_\ell^{(i)} \in \vartheta^{(ij)}} \left[\frac{x+p_\ell^{(i)}}{p_\ell^{(i)}} e_\ell^{(i)} \right] + \zeta \\
&\quad + \sum_{\tau_\ell^{(i)} \in \tau^{(ij)}} \frac{x+\phi+p_\ell^{(i)}}{p_\ell^{(i)}} e_\ell^{(i)} + \sum_{\tau_\ell^{(i)} \in \alpha^{(ij)}} \frac{x+p_\ell^{(i)}}{p_\ell^{(i)}} e_\ell^{(i)} - \beta^i_{\mathsf{prior}}(x) \\
&\quad \text{(Dropping subtracted values in the first and third terms)} \\
\Rightarrow 0 &< x u^{(i)} + \phi u^{(ij)} + u^{(i)} \max\left\{ p_\ell^{(i)} \right\} - x \frac{\Theta^{(i)}}{\Pi^{(i)}} + \frac{\Theta^{(i)}}{\Pi^{(i)}} (\Pi^{(i)} - \Theta^{(i)}) - \xi + \zeta \\
&\quad \text{(By Corollary 2)}
\end{aligned}
$$

Solving for x and noting that x must be an integer implies the upper bound of Equation 6.23. ■

We obtain upper bound for the conditions 6.20c, 6.20a and 6.20d in Lemmas 16, 18, and 18 correspondingly.

Lemma 16 *For any distinct modes $M^{(i)}$ and $M^{(j)}$, and $\beta \overset{\text{def}}{=} \beta^{i,j}_{\mathsf{trans}}(\delta^{(ij)})$, if Equation 6.20*c *of Theorem 3 is violated, then the violation must occur for some t at or before the maximum of $d^{(j)}_{\max}$ and the minimum of $\mathsf{lcm}\left\{ \{p_\ell^{(j)}\}_{\tau_\ell^{(j)} \in \tau^{(j)}} \cup \{\Pi^{(i)}\} \right\} + d^{(j)}_{\max}$ and*

$$
\left\lceil \frac{\sum_{\tau_\ell^{(i)} \in \tau^{(i)} \setminus \alpha^{(ij)}} e_\ell^{(i)} - \beta + \delta^{(ij)} u^{(ij)} + \sum_{\tau_\ell^{(j)} \in \tau^{(j)}} e_\ell^{(j)} + \frac{\Theta^{(j)}(\Delta^{(j)} - \Theta^{(j)})}{\Pi^{(j)}}}{\frac{\Theta^{(j)}}{\Pi^{(j)}} - u^{(j)}} \right\rceil . \tag{6.24}
$$

Proof: This lemma is proved using the same technique that we used for Lemma 15. The proof for the first part is skipped as it is same the previous one. The value that can be returned by $\mathsf{cidbf}(M^{(i)}, M^{(j)}, \delta^{(ij)} + t)$ is upper bounded by the non-aborted jobs of $M^{(i)}$. Let us assume that Equation 6.20c is violated for some t. Therefore,

$$\begin{aligned}
\beta^{i,j}_{\mathsf{post}}(0,t) \quad & < \sum_{\tau^{(i)}_\ell \in \tau^{(i)} \setminus \{\tau^{(ij)} \cup \alpha^{(ij)}\}} e^{(i)}_\ell \\
& + \sum_{\tau^{(ij)}_\ell \in \tau^{(ij)}} \left\lfloor \frac{t+\delta^{(ij)}}{p^{ij}_\ell} + 1 \right\rfloor e^{ij}_\ell \\
& + \sum_{\tau^{(j)}_\ell \in \tau^{(j)} \setminus \tau^{(ij)}} \mathsf{dbf}(\tau^{(j)}_\ell, t) \\
& - \beta^{i,j}_{\mathsf{trans}}(\delta^{(ij)}) \\
\Rightarrow \beta^{i,j}_{\mathsf{post}}(0,t) \quad & < \sum_{\tau^{(i)}_\ell \in \tau^{(i)} \setminus \{\tau^{(ij)} \cup \alpha^{(ij)}\}} e^{(i)}_\ell \\
& + \sum_{\tau^{(j)}_\ell \in \tau^{(j)} \setminus \tau^{(ij)}} \frac{t - d^{(j)}_\ell + p^{(i)}_\ell}{p^{(i)}_\ell} e^{(i)}_\ell \\
& + \sum_{\tau^{(ij)}_\ell \in \tau^{(ij)}} e^{ij}_\ell + (t + \delta^{(ij)}) U(\tau^{(ij)}) \\
& - \beta^{i,j}_{\mathsf{trans}}(\delta^{(ij)})
\end{aligned}$$

$$\begin{aligned}
\Rightarrow t\frac{\Theta^{(j)}}{\Pi^{(j)}} - \left[\frac{\Theta^{(j)}}{\Pi^{(j)}}\left(\Delta^{(j)} - \Theta^{(j)}\right)\right] & \\
< \sum_{\tau^{(i)}_\ell \in \tau^{(i)} \setminus \alpha^{(ij)}} e^{(i)}_\ell - \beta^{i,j}_{\mathsf{trans}}(\delta^{(ij)}) & \\
+ (t + \delta^{(ij)}) U(\tau^{(ij)}) + \sum_{\tau^{(j)}_\ell \in \tau^{(j)}} e^{(j)}_\ell &
\end{aligned}$$

(By Corollary 2)

Therefore, t must be less or equal to

$$\left\lceil \frac{\sum_{\tau^{(i)}_\ell \in \tau^{(i)} \setminus \alpha^{(ij)}} e^{(i)}_\ell - \beta + \delta^{(ij)} U(\tau^{(ij)}) + \sum_{\tau^{(j)}_\ell \in \tau^{(j)}} e^{(j)}_\ell + \frac{\Theta^{(j)}(\Delta^{(j)} - \Theta^{(j)})}{\Pi^{(j)}}}{\frac{\Theta^{(j)}}{\Pi^{(j)}} - u^{(j)}} \right\rceil$$

■

Lemma 17 *For any distinct modes $M^{(j)}$, if Equation 6.20*a *of Theorem 3 is violated, then the violation must occur for some t at or before the maximum of $d^{(j)}_{\max}$ and the minimum of* $\mathsf{lcm}\left\{\{p^{(j)}_\ell\}_{\tau^{(j)}_\ell\in\tau^{(j)}} \cup \{\Pi^{(j)}\}\right\} + d^{(j)}_{\max}$ *and*

$$\left\lceil \frac{u^{(j)} \times \max_{\tau^{(j)}_\ell\in\tau^{(j)}} \left\{p^{(j)}_\ell - d^{(j)}_\ell\right\} + \frac{\Theta^{(j)}(\Pi^{(j)}+\Delta^{(j)}-2\Theta^{(j)})}{\Pi^{(j)}}}{\frac{\Theta^{(j)}}{\Pi^{(j)}} - u^{(j)}} \right\rceil . \tag{6.25}$$

Proof: For proving the lemma, we use the very similar techniques to Lemma 15 and 16. Let say, Equation 6.20a is violated for some t. Therefore,

$$\begin{aligned}
0 &< \sum_{\tau^{(j)}_\ell\in\tau^{(j)}} \mathsf{dbf}(\tau^{(j)}_\ell, t) - \mathsf{sbf}(\Omega^{(j)}, t) \\
\Rightarrow 0 &< \sum_{\tau^{(j)}_\ell\in\tau^{(j)}} \left\lfloor \frac{t-d^{(j)}_\ell}{p^{(j)}_\ell} + 1 \right\rfloor \times e^{(j)}_\ell \\
&\quad - \left[t \times \tfrac{\Theta^{(j)}}{\Pi^{(j)}} - \tfrac{\Theta^{(j)}}{\Pi^{(j)}}(\Pi^{(j)} + \Delta^{(j)} - 2\Theta^{(j)})\right] \\
\Rightarrow 0 &< \sum_{\tau^{(j)}_\ell\in\tau^{(j)}} \frac{t-d^{(j)}_\ell+p^{(j)}_\ell}{p^{(j)}_\ell} \times e^{(j)}_\ell \\
&\quad - \left[t \times \tfrac{\Theta^{(j)}}{\Pi^{(j)}} - \tfrac{\Theta^{(j)}}{\Pi^{(j)}}(\Pi^{(j)} + \Delta^{(j)} - 2\Theta^{(j)})\right] \\
\Rightarrow 0 &< t \times u^{(j)} + \max_{\tau^{(j)}_\ell\in\tau^{(j)}} \left\{p^{(j)}_\ell - d^{(j)}_\ell\right\} \times u^{(j)} \\
&\quad - \left[t \times \tfrac{\Theta^{(j)}}{\Pi^{(j)}} - \tfrac{\Theta^{(j)}}{\Pi^{(j)}}(\Pi^{(j)} + \Delta^{(j)} - 2\Theta^{(j)})\right] \\
\Rightarrow t &< \frac{u^{(j)} \times \max_{\tau^{(j)}_\ell\in\tau^{(j)}} \left\{p^{(j)}_\ell - d^{(j)}_\ell\right\} + \frac{\Theta^{(j)}(\Pi^{(j)}+\Delta^{(j)}-2\Theta^{(j)})}{\Pi^{(j)}}}{\frac{\Theta^{(j)}}{\Pi^{(j)}} - u^{(j)}}
\end{aligned}$$

Above equations imply the Lemma. ■

Lemma 18 *For any distinct mode change from $M^{(i)}$ to $M^{(j)}$ and integer $s : 0 < s \leq \delta^{(ij)}$, if Equation 6.20*d *of Theorem 3 is violated, then the violation must occur for some t at or before* *the maximum of* $d^{(j)}_{\max}$ *and the minimum of* $\mathsf{lcm}\left\{\{p^{(j)}_\ell\}_{\tau^{(j)}_\ell\in\tau^{(j)}} \cup \{\Pi^{(j)}\}\right\} + d^{(j)}_{\max}$ *and*

$$\left\lceil \frac{s\cdot u^{(ij)} + u^{(j)} \times \max\limits_{\tau^{(j)}_\ell\in\tau^{(j)}}\left\{p^{(j)}_\ell - d^{(j)}_\ell\right\} - \frac{s\cdot\Theta^{(ij)}}{\Pi^{(ij)}} + \frac{\Theta^{(j)}(\Delta^{(j)}-\Theta^{(j)})}{\Pi^{(j)}} + \frac{\Theta^{(ij)}(\Pi^{(ij)}-\Theta^{(ij)})}{\Pi^{(ij)}}}{\frac{\Theta^{(j)}}{\Pi^{(j)}} - u^{(j)}} \right\rceil. \quad (6.26)$$

Proof: This lemma is proved using the same technique that we used for Lemma 16. Let us assume that Equation 6.20d is violated for some t. Therefore,

$$\begin{aligned}
&\beta^{i,j}_{\mathsf{post}}(s,t)\\
&\qquad < \sum_{\tau^{(j)}_\ell\in\tau^{(j)}\setminus\tau^{(ij)}} \mathsf{dbf}(\tau^{(j)}_\ell, t) \sum_{\tau^{(ij)}_\ell\in\tau^{(ij)}} \mathsf{dbf}(\tau^{(ij)}_\ell, t+s)\\
&\Rightarrow \beta^{i,j}_{\mathsf{post}}(s,t)\\
&\qquad < \sum_{\tau^{(j)}_\ell\in\tau^{(j)}\setminus\tau^{(ij)}} \frac{t-d^{(j)}_\ell+p^{(i)}_\ell}{p^{(i)}_\ell} e^{(i)}_\ell + \sum_{\tau^{(ij)}_\ell\in\tau^{(ij)}} \frac{t+s-d^{(ij)}_\ell+p^{(ij)}_\ell}{p^{(ij)}_\ell} e^{(ij)}_\ell\\
&\Rightarrow t\frac{\Theta^{(j)}}{\Pi^{(j)}} - \left[\frac{\Theta^{(j)}}{\Pi^{(j)}}\left(\Delta^{(j)} - \Theta^{(j)}\right)\right] + s\frac{\Theta^{(ij)}}{\Pi^{(ij)}} - \left[\frac{\Theta ij}{\Pi^{(ij)}}\left(\Pi^{(ij)} - \Theta^{(ij)}\right)\right]\\
&\qquad < s\cdot U(\tau^{(ij)}) + \left(t + \max\nolimits_{\tau^{(j)}_\ell\in\tau^{(j)}}\left\{p^{(j)}_\ell - d^{(j)}_\ell\right\}\right)\cdot U(\tau^{(i)}).
\end{aligned}$$

Solving the above for t implies the lemma. ■

Finally, the following corollary on the number of iterations for convergence of ci follows immediately from Equation 6.18 of Theorem 3.

Corollary 7 *The convergence of the sequence defined in Equation 6.19 occurs at* $g \le \sum\limits_{i\le q} \max\limits_{\substack{j\in[1,q]\\ i\ne i}} \mathsf{ci}^{(ij)}$ *where* $\mathsf{ci}^{(ij)}$ *is the value defined in Equation 6.18.*

Proof: This lemma is established using the property of $\mathsf{cidbf}(M^{(i)}, M^{(j)}, \phi)$. Equation 6.19 calculates ci iteratively and this equation is monotonically increasing (see Lemma 13). The value at g-th iteration determined from that of $(g-1)$-th iteration. Therefore, for getting a different value of ci in future iterations, ci value for at least one mode needs to be changed. The maximum carry-in demand that a mode $M^{(i)}$ can generate is $\max_{j=1,\ldots,q\wedge i\ne i} \mathsf{ci}^{(ij)}$. The cidbf function is monotonically non-decreasing, and integer valued function. Therefore, the maximum steps required for calculating carry-in demand are finite and can be determined using the sum of the maximum values for each pair of modes for all steps. ■

6.3 Algorithms

In this section, we join all the pieces together to develop a schedulability test algorithm for non-concrete MCRs. First we construct the algorithm MaxCarry that calculates the maximum carry-in for each pair of modes $M^{(i)}$ and $M^{(j)}$. We then present our algorithm called *schedulability using bounded iteration* (SUBI) for checking the schedulability of any multi-modal subsystem $\mathcal{S}$ with non-concrete MCRs. (Please note, that since Theorem 3 covers all possible sequence of MCRs, any subsystem that satisfies Theorem 3 will also satisfy Theorem 2 for any fixed legal concrete sequence of MCRs.)

§**Maximum Carry.** The algorithm MaxCarry (pseudocode shown in Algorithm 1) obtains an upper bound on the maximum carry-in for each pair of modes $M^{(i)}$ and $M^{(j)}$. In each step MaxCarry calculates a new upper bound for carry-in based on the upper bound obtained in previous iterations. The iteration continues until the carry-in bound for all pairs of modes is unchanged from the previous iteration. The algorithm converges after a finite number of steps which is established in Corollary 7. Let P denote $\max_{i,j}\{\max(\delta^{(ij)} + d_{\max}^{(ij)}, d_{\max}^{(i)})\}$. The runtime complexity of each iteration depends on the $\widetilde{\Psi}$ function. The function $\widetilde{\Psi}$ given ζ determines carry-in demand using $O\left(n \times Q(P, B)\right)$ steps where n is the maximum total number of tasks in any mode and B is the maximum value of Equation 6.18 and $Q(\phi, \zeta)$ is Equation 6.23 as a function of ϕ and ζ. Therefore, the time complexity of MaxCarry is $O\left(nq^2BQ(P, B)\right)$. B is obviously polynomial in the mode parameters (and thus pseudo-polynomial in the representation). When $\frac{\Theta^{(i)}}{\Pi^{(i)}} - u^{(i)}$ for all $M^{(i)}$ is lower bounded by a fixed positive constant, $Q(\phi, \zeta)$ is a pseudo-polynomial function; thus, the total time complexity is pseudo-polynomial. (Note that a pseudo-polynomial function given a pseudo-polynomial input returns a pseudo-polynomial value.)

§**Schedulability Algorithm.** The algorithm SUBI (presented in Algorithm 2) checks for schedulability for non-concrete sequences of MCRs. This algorithm uses MaxCarry as a subroutine.

Algorithm 1 $\mathsf{MaxCarry}(\mathcal{S})$.

```
1:  {Returns a $[q \times q]$ matrix $\zeta$.}
2:  $\zeta \Leftarrow 0$
3:  **repeat**
4:      change $\Leftarrow false$
5:      **for** $i = 1$ to $q$ **do**
6:          $\mathsf{ci}^{i}_{\max} \leftarrow \max_{h=1,\ldots,q \wedge h \neq i}\{\zeta_{hi}\}$
7:          **for** $j = 1$ to $q$ **do**
8:              $W^{(ij)}$ is defined by Equation 6.18.
9:              $d^{i}_{max} \Leftarrow \max\left(\delta^{(ij)} + d^{(ij)}_{\max}, d^{(i)}_{\max}\right)$
10:             $\mathsf{ci} \Leftarrow \widetilde{\Psi}\left(M^{(i)}, M^{(j)}, d^{i}_{max}, \mathsf{ci}^{i}_{\max}\right) - \beta^{i,j}_{\mathsf{trans}}(\delta^{(ij)})$
11:                 $- \sum_{\tau^{(ij)}_{\ell} \in \tau^{(ij)}} \mathsf{dbf}\left(\tau^{(ij)}_{\ell}, d^{i}_{max} - \delta^{(ij)} - p^{(ij)}_{\ell}\right)$
12:             **if** $\min(\mathsf{ci}, W^{(ij)}) > \zeta^{(ij)}$ **then**
13:                 $\zeta^{(ij)} \Leftarrow \min(\mathsf{ci}, W^{(ij)})$
14:                 change $\Leftarrow true$
15:             **end if**
16:         **end for**
17:     **end for**
18: **until** change $= false$
19: **return** $\zeta$
```

The algorithm checks all five conditions of Theorem 3 for schedulability. The *for* loop at Line 1 uses the condition of Equation 6.20a, the loop at Line 10 checks the condition of Equation 6.20b, and the innermost loop starting at Line 27 checks the condition of Equation 6.20c. MaxCarry is called prior to checking the conditions of Equations 6.20b and 6.20c, so that the maximum carry-in can be used from the stored value. Equations 6.20d and 6.20e are checked in the second main loop. The algorithm returns true only if all of the above mentioned five conditions do not fail for any interval length of t.

Let R correspond to the maximum value of Equation 6.25 of Lemma 17, S correspond to the maximum value of Equation 6.24 of Lemma 16 over all modes, $V(s)$ correspond to the maximum value of Equation 6.26 of Lemma 18 (given a value of s), and $\delta_{\max} \overset{\text{def}}{=} \max_{i,j\in\{1,\ldots,q\}}\{\delta^{(ij)}\}$. Observe that the maximum value for s in Lemma 18 is $\delta_{\max}$. The first loop requires $O(nqR)$ steps; MaxCarry requires $O\left(nq^2 BQ(P,B)\right)$; and the second loop requires $O\left(nq^2 \left(P \times Q(P,B)\right. +P\times\right.$

Algorithm 2 $\mathsf{SUBI}(\mathcal{S})$.

```
1:  for $i = 1$ to $q$ do
2:      $\mathcal{T}$ is set by Equation 6.25.
3:      for $t = 1$ to $\mathcal{T}$ do
4:          if $\mathsf{sbf}(\Pi^{(i)}, \Theta^{(i)}, t) < \mathsf{dbf}(\tau^{(i)}, t)$ then
5:              return $false$
6:          end if
7:      end for
8:  end for
9:  $\zeta \Leftarrow \mathsf{MaxCarry}(\mathcal{S})$
10: for $i = 1$ to $q$ do
11:     for $j = 1$ to $q$ do
12:         for $\phi = 0$ to $\delta^{(ij)}$ do
13:             if $\widetilde{\Psi}(M^{(i)}, M^{(j)}, \phi, \zeta^{(ij)}) > \beta^{i,j}_{\mathsf{trans}}(\phi)$ then
14:                 return $false$
15:             end if
16:             if $\mathsf{dbf}(\tau^{(ij)}, \phi) \leq \mathsf{sbf}(\Omega_{(ij)}, \phi)$ then
17:                 return $false$
18:             end if
19:             $\mathcal{T}$ is set by Equation 6.26
20:             for $t = 0$ to $\mathcal{T}$ do
21:                 if Equation 6.20d is false for $s = \phi$ then
22:                     return $false$
23:                 end if
24:             end for
25:         end for
26:         $\mathcal{T}$ is set by Equation 6.24.
27:         for $t = 0$ to $\mathcal{T}$ do
28:             $\mathsf{carry} \Leftarrow \widetilde{\Psi}(M^{(i)}, M^{(j)}, \delta^{(ij)} + t, \zeta^{(ij)})$
29:                         $-\beta^{i,j}_{\mathsf{trans}}(\delta^{(ij)})$
30:             if $\mathsf{carry} + \mathsf{dbf}(\tau^{(i)} \setminus \tau^{(ij)}, t) > \beta^{i,j}_{\mathsf{post}}(0, t)$ then
31:                 return $false$
32:             end if
33:         end for
34:     end for
35: end for
36: return $true$
```

$V(\delta_{\max}) + S \times Q(P+S, B)$. Thus, the total runtime is again pseudo-polynomial-time complexity if $\frac{\Theta^{(i)}}{\Pi^{(i)}} - u^{(i)}$ is lower bounded by a fixed positive constant for all $M^{(i)}$

6.4 Simulations

In this section, we present the performance results for our proposed algorithm. We compare SUBI with exponential-time *schedulability analysis using reachability graph* (SURG) proposed by Phan et al. [63]. For the simulation, we implemented SURG and SUBI in MATLAB and performed our simulations on a 2.33GHz Intel Core 2 Duo machine with 2.0GB RAM. During the simulation, we have the following parameters and value ranges for the multi-modal subsystem $\mathcal{S}$:

Tasks	Properties			$Modes$		
#	$e^{(i)}$	$d^{(i)}$	$p^{(i)}$	$M^{(1)}$	$M^{(2)}$	$M^{(3)}$
1	1	10	10	y	y	y
2	3	30	30	y	y	n
3	4	40	40	n	n	y
4	1	10	10	y	n	n
5	1	20	20	n	y	y
6	3	24	24	n	n	y
7	2	20	20	y	n	n
8	1	10	10	n	y	n

Table 6.1: EDF Schedulability Comparison: Tasks Properties.

1. The number of modes (q) in the subsystem is 3.
2. The total number of tasks in the multi-modal tasks system is 8. Task properties and distributions are described in Table 8.1.
3. During a mode transition, jobs from task τ_1 are considered as aborted jobs. Tasks τ_5 is unchanged between MCRs involving M_2/M_3.
4. The resource period (Π) and deadline (Δ) are set to 10 for all modes.
5. The offset $\delta^{(ij)}$ is set to Π for both simulations. $N^{(i)}$ is set equal to 2 for all modes $M^{(i)}$.
6. A 'y' in the mode column indicates this task is present in the mode.

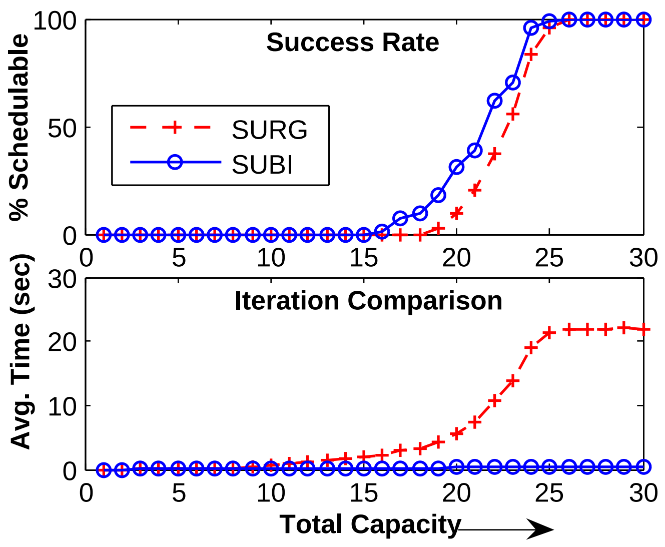

Figure 6.8: EDF Schedulability Comparison: Efficacy of SUBI vs SURG.

In the first simulation, we randomly generate a set of capacities ($\Theta^{(i)}$) where the total sum is taken from the range $[1, q\Pi]$. We execute SUBI and SURG for checking schedulability of the subsystem $\mathcal{S}$ with R. The graph at the top of the Figure 6.8 presents the percentage of 'YES' responses out of 200 run on each distinct summation of capacities (i.e., the value on the x axis). The dashed line depicts the results for SURG, and solid line is for SUBI. The graph at the bottom in Figure 6.8 presents the average elapsed time for deciding the schedulability using over randomly-generated capacities for each given utilization. For this particular subsystem, Figure 6.8 illustrates that SUBI does as well or better than SURG and is clearly more efficient.

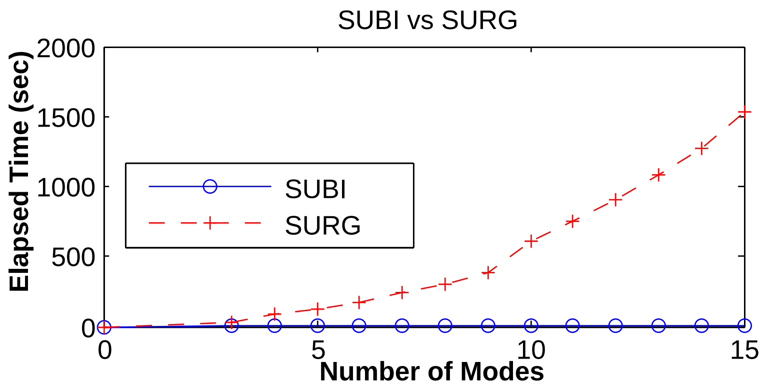

Figure 6.9: EDF Schedulability Comparison: Scalability of SUBI vs SURG

For checking the scalability of SUBI, we perform a second simulation with a higher number of modes (up to 15). In each step, we increase by one mode with four tasks chosen randomly from Table 8.1 and perform schedulability test using SURG and SUBI with capacity equals to the highest value (i.e., Π). The result is depicted in Figure 6.9. The dashed line with plus markers shows the elapsed time in second for reaching a decision using SURG, whereas the solid line depicts the results for SUBI. SURG is more general, and designed to compute the feasible minimum capacity using reachability graph; thus, the higher running time of SURG is due to the exponential-time complexity of traversing reachability graph.

6.5 Conclusion

In this chapter, we derived a EDF schedulability test for mode-change request for two settings: concrete and non-concrete mode change requests. For non-concrete subsystems (i.e., the sequence of mode-changes are not known a priori), we obtain a schedulability analysis algorithm that has pseudo-polynomial time complexity. The previous known algorithm which uses a reachability graph requires exponential time complexity. Furthermore, our simulation results validate the effectiveness and efficiency of algorithm and demonstrate that it scales as the number of modes increases. Thus, our proposed approach can be used to quickly verify the schedulability of control systems with a large number of modes. In the next chapter, we develop schedulability analysis for FP schedulability analysis does not apply readily to FP-scheduled or non-preemptive multi-modal systems. Later, we accelerate the schedulability analysis using parallel platform so that these schedulability analysis can utilized for design-space exploration.

CHAPTER 7: FP SCHEDULABILITY OF MMS

In the previous chapter, we developed the analysis for EDF scheduled multi-modal systems. While EDF is an optimal scheduling algorithm (if there is a possible schedule for a set of jobs that meet all deadlines, the schedule generated by EDF will also maintain all the timing constraints), the implementation requires additional data-structures for accounting timing requirements at job level. On the other hand, fixed-priority scheduling (e.g., rate monotonic, deadline monotonic) are simpler and more commonly used in industrial automation, automotive control systems, and power plants. Therefore, in this dissertation, we also present schedulability analysis for a multi-modal system that schedule jobs using fixed-priority (FP) algorithm upon a non-continuous processing platform modeled by periodic resources [32].

In addition, real-time systems that interacts with the physical environment (i.e., cyber-physical systems) place additional constraints upon software aspects of the system. For instance, sensing and actuation often require non-preemption to ensure correct data acquisition (e.g., an ultrasonic sensor in a robotic car). However, none of the existing fixed-priority schedulability analysis for multi-modal systems (e.g., [63, 74]) can address non-preemptible resource access. We consider non-preemptible execution only with FP scheduled multi-modal systems as this issue is mostly found in industrial automation where FP is prevalent. Non-preemptivity with EDF scheduled multi-modal systems is also an interesting research problem which is left for future research.

In this chapter, we propose the fixed-priority schedulability analysis of multi-modal CPS with strict timing requirements (hard real-time constraints). We emphasize the characteristics (e.g., the minimum separation between successive mode-changes) of the multimode model by Hettiarachchi et al. [42] for a thermal-aware processor control system where the software is specified by a sporadic task system [51] and the hardware is represented by the periodic resource model [32, 70]. We develop a pseudo-polynomial schedulability analysis for the multi-modal systems. A case-study of radar-assisted cruise control system is included to show the usability of multimode system which consists of fixed priority non-preemptive tasks. For comparison, we perform additional simulations which indicate that our algorithm achieves better efficiency over the state-of-the-art [63] with no loss of schedulability. In the next section, we first show the typical unimodal schedulability may produce wrong results for multi-modal systems.

7.1 Invalidity of Standard FP Critical Instant Theorem

Previous work on multi-modal system using FP assumed dedicated processing resource for the application. For systems that can change hardware modes along with software modes, the critical instant theorem does not hold as shown in the Figure 7.1. In Figure 7.1, two modes are depicted where each mode specifies both software and its hardware requirements. Two unchanged tasks [$\tau_1^{(12)} \overset{\text{def}}{=} (1,4,4)$ and $\tau_2^{(12)} \overset{\text{def}}{=} (1,4,4)$] are common (forward diagonally hatched rectangles in Figure 7.1) in between these two modes. $M^{(2)}$ has one additional finished task $\tau_3^{(2)} \overset{\text{def}}{=} (1,10,10)$. The mode $M^{(1)}$ has the hardware requirement $(6,3,3)$, whereas the $M^{(2)}$ specifies the maximum hardware resource $(6,6,6)$. In the left-hand-side, the new-mode task is released with typical critical instant scenario and the response time is 4. At the bottom, vertically hatched rectangles

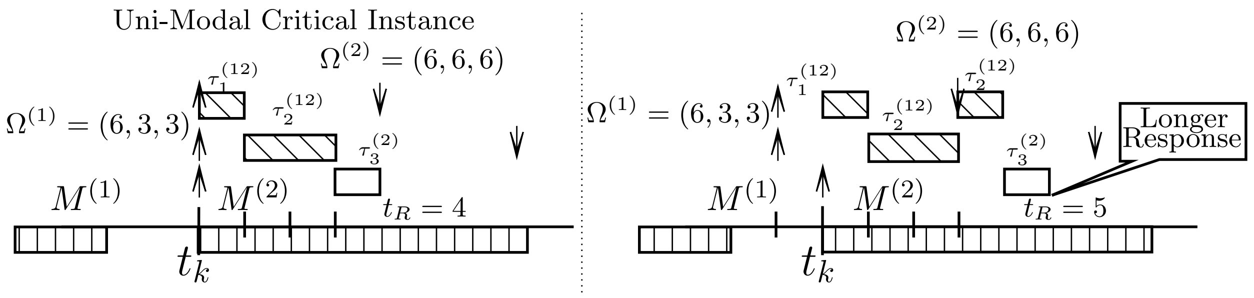

Figure 7.1: FP-Scheduled Multi-Modal System: Longer response time in presence of common tasks.

depict availability for these periodic resources. The right-hand-side figure depicts a scenario where common tasks are released 1 unit before the mode change request which eventually results longer response time (5). To address this issue, Stoimenov et al. [74] consider two jobs for unchanged tasks at transition which reduces the accuracy of the schedulability. Phan et al. [63] explore a reachability graph for schedulability, which may take exponential time. In contrast, we consider bounded busy-intervals to develop a pseudo-polynomial schedulability.

7.2 Tasks Priority

For priority of each task, we consider global priority ordering where the priority of a task $\tau_k^{(i)}$ is taken from a global fixed set of priorities $\wp \overset{\text{def}}{=} \{\wp^{(1)}, \wp^{(2)} \ldots\}$ such that $\wp^{(j)} \in \mathbb{N}^+$ for each $\wp^{(j)} \in \rho$. We also define the function : $M \times \tau \to \wp$ that takes a task and a mode (given in the superscript of the task), and returns the task's priority in the specified mode as follows:

$$\left((\tau_\ell^{(i)}) \geq (\tau_k^{(i)})\right) \Longleftrightarrow \left(\tau_\ell^{(i)} \preccurlyeq \tau_k^{(i)}\right) \tag{7.1}$$

and

$$\left((\tau_\ell^{(i)}) \geq (\tau_k^{(j)})\right) \Longleftrightarrow \left(\tau_\ell^{(i)} \preccurlyeq \tau_k^{(j)}\right) \tag{7.2}$$

To be consistent, all tasks in $\tau^{(ij)}$ must have the same priority values in both $\tau^{(i)}$ and $\tau^{(j)}$; otherwise, the task will be a member of $\vartheta^{(ij)}$. The function $\mathsf{hp}\,(A, \tau_\ell)$ is defined for $\tau_\ell \in \tau^{(i)}$ and a subset $A \subseteq \tau^{(i)}$ as $\{\tau_k \in A | \tau_\ell \preccurlyeq \tau_k\}$ (note $\mathsf{hp}\,(A, \tau_\ell)$ includes τ_ℓ). We also define the set of lower-priority tasks as $\mathsf{lp}\,(A, \tau_\ell) \overset{\text{def}}{=} A \backslash \mathsf{hp}\,(A, \tau_\ell)$. For flexibility, we overload both $\mathsf{hp}\,(A, \rho_v)$ and $\mathsf{lp}\,(A, \rho_v)$ with a priority value $\rho_v \in \rho$ as $\{\tau_\ell \in A | (\tau_\ell) \leq \rho_v\}$ and $A \setminus \mathsf{hp}\,(A, \rho_v)$ respectively. We assume uninterrupted execution for non-preemptive region of a task τ_ℓ within $\mathcal{M}$; therefore, the maximum lower-priority blocking for a task τ_ℓ due to non-preemptive execution of lower-priority tasks of the set A is denoted $B(A, \tau_\ell)$. $B(A, \tau_\ell)$ can be obtained from $\max_{\tau_k \in \mathsf{lp}(A, \tau_\ell)} \{\mathsf{ne}_k\}$.

7.3 FP Schedulability Analysis

Traditional schedulability analysis for unimodal systems differs from multi-modal systems, that support software/hardware modes, in ways the total requests and processing resource are quantified. To improve the accuracy of the schedulability analysis with less pessimism, characterizing the minimum supply with respect to a mode-change is important. In the previous chapter, we defined and quantified the minimum resource-execution supply function with respect to an MCR which are summarized as follows.

Definition 13 (mcr-sbfs) *For a mode-change request* $\mathsf{mcr}_k \overset{def}{=} (M^{(i)}, M^{(j)}, t_k)$*, the functions* $\beta^{i}_{\mathsf{prior}}(t)$*,* $\beta^{i,j}_{\mathsf{trans}}(t)$*, and* $\beta^{i,j}_{\mathsf{post}}(s,t)$ *quantify the minimum execution respectively prior to* mcr_k *(i.e.,* $[t_k - t, t_k]$*) guaranteed by* $\Omega^{(i)}$*, during the transition after* mcr_k *(i.e.,* $[t_k, t_k + t]$*) guaranteed by* $\Omega^{(ij)}$ *and* $\Omega^{(j)}$*, and beyond the transition after* mcr_k *(i.e.,* $[t_k + \delta_{ij} - s, t_k + \delta_{ij} + t]$*) guaranteed by* $\Omega^{(ij)}$ *and* $\Omega^{(j)}$ *where* $0 \le s \le \delta_{ij}$*. Upper bounds for these functions can be obtained as follows (derived in Section 6.1.2):*

$$\beta^{i}_{\mathsf{prior}}(t) \ge a\Theta^{(i)} + \min\Big(\Theta^{(i)}, \big(t - \big((a+1)\Pi^{(i)} - \Theta^{(i)}\big)\big)_+\Big) \tag{7.3}$$

$$\begin{aligned} \beta^{i,j}_{\mathsf{trans}}(t) \ge{}& b\Theta^{(ij)} + c\Theta^{(j)} \\ &+ \min\Big(\Theta^{(ij)}, \big(\min(t,\delta_{ij}) - \big(b\Pi^{(ij)} + \Delta^{(ij)} - \Theta^{(ij)}\big)\big)_+\Big) \\ &+ \min\Big(\Theta^{(j)}, \big((t-\delta_{ij})_+ - \big(c\Pi^{(j)} + \Delta^{(j)} - \Theta^{(j)}\big)\big)_+\Big) \end{aligned} \tag{7.4}$$

$$\begin{aligned} &\beta^{i,j}_{\mathsf{post}}(s,t) \ge \\ &\quad d\Theta^{(ij)} + \min\Big(\Theta^{(ij)}, \big(s - \big((d+1)\Pi^{(ij)} - \Theta^{(ij)}\big)\big)_+\Big) \\ &\quad + f\Theta^{(j)} + \min\Big(\Theta^{(j)}, \big(t - \big(f\Pi^{(j)} + \Delta^{(j)} - \Theta^{(j)}\big)\big)_+\Big) \end{aligned} \tag{7.5}$$

where $a \overset{\text{def}}{=} \left\lfloor \frac{t}{\Pi^{(i)}} \right\rfloor$, $b \overset{\text{def}}{=} \left\lfloor \frac{\min(t,\delta_{ij})}{\Pi^{(ij)}} \right\rfloor$, $c \overset{\text{def}}{=} \left\lfloor \frac{(t-\delta_{ij})_+}{\Pi^{(j)}} \right\rfloor$, $d \overset{\text{def}}{=} \left\lfloor \frac{s}{\Pi^{(ij)}} \right\rfloor$, and $f \overset{\text{def}}{=} \left\lfloor \frac{t}{\Pi^{(j)}} \right\rfloor$.

We now derive a schedulability test following the same framework as the derivations of sufficient schedulability analysis for unimodal systems. We utilize both response-time and request-supply analysis. For response-time analysis, we adapt the strategy developed by Davis et al. [29]. For request/supply analysis, we obtain *conditions to miss a deadline while scheduled by* FP. Next, we take the contrapositive of these conditions to obtain schedulability tests for multi-modal systems.

A deadline miss may occur due to intra-mode issues (e.g. insufficient resources), or inter-modes issues (e.g., remaining execution from a mode change request). To generalize the analysis, we define following terms for different types of deadline misses. Assuming the system is not schedulable, we consider the following scenario for schedulability analysis:

> Deadline-Miss-Event (DME): The subsystem misses the first deadline after the k-th mode change request $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$. Suppose that $t_d < t_{k+1}$ denotes the earliest instant at which the FP-schedule misses a deadline for a job ($\tau_{f,d}$) of task τ_f. Let t_s ($< t_d$) denote the latest time-instant before t_d that the system does not have any active job of higher or equal priority than that of τ_f.

We address the interval $[t_s, t_d]$ as a *busy-interval*. Based on the distribution of $[t_s, t_d]$ with respect to mcr_k, the tasks/resources that contribute to request/supply could be different (Figure 7.2).

- BI_1: contained by the execution of a single mode $M^{(j)}$ (i.e., $t_k + \delta_{ij} \le t_s, t_d \le t_{k+1}$).
- BI_2: contained by a transition interval past mcr_k (i.e., $t_k \le t_s \le t_k + \delta_{ij}$ and $t_k \le t_d \le t_k + \delta_{ij}$).
- BI_3: starts at the transition period, but ends in $M^{(j)}$ (i.e., $t_k \le t_s < t_k + \delta_{ij}$ and $t_k + \delta_{ij} < t_d \le t_{k+1}$).
- BI_4: starts prior to t_k (i.e., prior to $M^{(j)}$ of mcr_k) and ends during transition (i.e., $t_s < t_k < t_d \le t_k + \delta_{ij}$).
- BI_5: starts prior to t_k (i.e., in any mode prior to $M^{(j)}$ of mcr_k) and ends in $M^{(j)}$ (i.e., $t_s < t_k, t_k + \delta_{ij} < t_d \le t_{k+1}$).

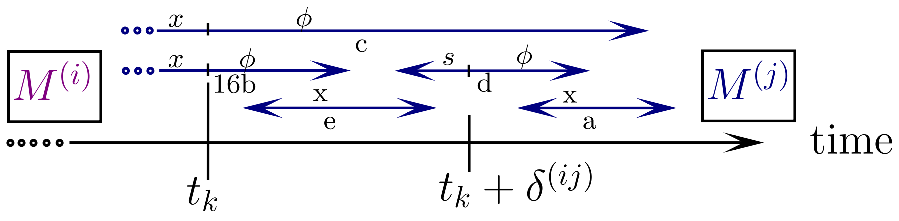

Figure 7.2: Busy intervals: arrows depict possible distributions of $[t_s, t_d]$

We assume J is the set of higher or equal priority jobs that are active during the interval $[t_s, t_d]$. The jobs of J may be blocked by non-preemptive execution of a lower-priority job; however, this priority-inversion may occur once at the start of $[t_s, t_d]$. We denote this lower-priority blocking as b. We also assume that $\eta(J, t_s, t_i)$ denotes the total execution requests in the interval $[t_s, t_i]$ for the job set J. As there is a missed deadline at t_d and the processor was continuously busy in $[t_s, t_d]$, the following must be true for J:

$$\forall t_i \in [t_s, t_d] : \quad b + \eta(J, t_s, t_i) > \beta(t_s, t_i) \tag{7.6}$$

where $\beta(t_s, t_i)$ is the supply received for the interval $[t_s, t_i]$. As we are considering non-preemptive systems, there may be multiple jobs from the task with the missed deadline. Based on interval distributions, the sets of tasks that constitute $\eta(J, t_s, t)$ are different. We start with BI_1, BI_2, and BI_3 as the analysis is similar to unimodal systems, which gradually develops the concepts for the analysis with carry-in.

7.3.1 Individual Mode Schedulability

For BI_1, BI_2, and BI_3, higher-priority old-mode jobs must be finished before t_s (by definition of busy interval). In these cases, the tasks that constitute $\eta(J, t_s, t_i)$ are a subset of $M^{(j)}$. In the next three lemmas, we develop necessary conditions of deadline misses for busy intervals of type BI_1, BI_2, and BI_3.

With BI_1 and BI_2, the task set remains the same (i.e., $\tau^{(j)}$ and $\tau^{(ij)}$, respectively) for the entire busy interval; however, the analysis by Davis et al. [29] to address non-preemptivity will not apply directly due to non-continuous execution of periodic resources. We define the largest busy-interval and response time in the following theorem. We assume the active taskset $\tau^{(f)}$ could be either $\tau^{(j)}$ or $\tau^{(ij)}$ for BI_1 or BI_2.

Theorem 4 *A critical instant, with type either BI_1 or BI_2, for a job of task $\tau_\ell^{(f)} \in \tau^{(f)}$ may occur when one of its jobs is released with higher-priority jobs immediately after the processing resource starts executing the largest non-preemptive task in $\tau^{(f)}$.*

The proof for the preceding theorem is skipped as it follows from unimodal schedulability. For the largest busy-interval, Davis et al. [29] considered continuous resource which will not apply for periodic resources. We extend the analysis as follows:

For a MM system, the largest busy-interval $\mathrm{L}_\ell^{(f)}$ for a task $\tau_\ell^{(f)} \in \tau^{(f)}$ with BI_1 and BI_2 depends on both request and supply as follows where $m \in \mathbb{N}$:

$$\begin{aligned}
&\mathrm{L}_{\ell,0}^{(f)} \overset{\text{def}}{=} B(\tau^{(f)}, \tau_\ell^{(f)}) + e_\ell^{(f)} && 7.7a \\
&\bar{\mathrm{L}}_{\ell,m}^{(f)} \overset{\text{def}}{=} \left\{ B(\tau^{(f)}, \tau_\ell^{(f)}) + \sum_{\tau_k^{(f)} \in \mathsf{hp}\left(\tau_\ell^{(f)}, \tau^{(f)}\right)} \left\lceil \frac{\mathrm{L}_{\ell,m}^{(f)}}{p_k^{(f)}} \right\rceil \times e_k^{(f)} \right\} && 7.7b \\
&\mathrm{L}_{\ell,m+1}^{(f)} \overset{\text{def}}{=} \mathsf{na} + \left\lfloor \frac{\bar{\mathrm{L}}_{\ell,m}^{(f)}}{\Theta^{(f)}} \right\rfloor \times \Pi^{(f)} + \mu_{\geq 0}(\mathrm{ps}) \cdot [\Pi^{(f)} - \Theta^{(f)} + \mathrm{ps}] && 7.7c
\end{aligned} \tag{7.7}$$

where $\mathsf{na} \overset{\text{def}}{=} \Pi^{(f)} + \Delta^{(f)} - 2\Theta^{(f)}$ and $\mathrm{ps} \overset{\text{def}}{=} \bar{\mathrm{L}}_{\ell,m}^{(f)} - \left\lfloor \frac{\bar{\mathrm{L}}_{\ell,m}^{(f)}}{\Theta^{(f)}} \right\rfloor \times \Theta^{(f)}$. $\mathrm{L}_\ell^{(f)}$ takes the value of $\mathrm{L}_{\ell,m}^{(f)}$ at the convergence for the minimum m such that $\mathrm{L}_{\ell,m}^{(f)} = \mathrm{L}_{\ell,m+1}^{(f)}$.

The recurrence initializes a busy-interval length with the maximum blocking time and the execution time for a task $\tau_\ell^{(f)}$ in Equation 7.7a. Equation 7.7b quantifies the total requests in a busy-interval taken multiple jobs into account. As mentioned earlier, traditional response time analysis is not sufficient due to the assumption of continuous resource execution; therefore, Equation 7.7c accounts for supply to complete the request calculated in Equation 7.7b. First part na of Equation 7.7c is due to the maximum resource unavailability at the beginning of a mode. The last part is due to the fraction of the resource cycle required for $\bar{\mathrm{L}}_{\ell,m}^{(f)} - \left\lfloor \frac{\bar{\mathrm{L}}_{\ell,m}^{(f)}}{\Theta^{(f)}} \right\rfloor \times \Theta^{(f)}$.

As shown by Davis et al. [29], the largest response time for non-preemptive setting is no longer for the first job in case of synchronous arrival. To be schedulable, we need to make sure that all the jobs meet their deadlines. We consider in total $Y = \lceil \mathrm{L}_\ell^{(f)} / p_\ell^{(f)} \rceil$ numbers of jobs. The response time for y-th job is denoted by $R_\ell^{(f)}(y)$. So, for any task $\tau_\ell^{(f)} \in \tau^{(f)}$, the response time $R_\ell^{(f)} \stackrel{\text{def}}{=} \max_{y=1\ldots Y}\{R_\ell^{(f)}(y)\}$ must be less than $d_\ell^{(f)}$. To calculate $R_\ell^{(f)}(y)$, we adapt an iterative approach as follows ($k \in \mathbb{N}$)

$$\begin{aligned} R_{\ell,0}^{(f)}(y) &\stackrel{\text{def}}{=} B(\tau^{(f)}, \tau_\ell^{(f)}) + (y-1) \times e_\ell^{(f)} \\ \bar{R}_{\ell,k}^{(f)}(y) &\stackrel{\text{def}}{=} R_{\ell,0}^{(f)}(y) + \sum_{\tau_c^{(f)} \in \mathsf{hp}\left(\tau^{(f)} \setminus \tau_\ell^{(f)}, \tau_\ell^{(f)}\right)} \left\lceil \frac{R_{\ell,k}^{(f)}(y)}{p_c^{(f)}} \right\rceil \times e_c^{(f)} \\ R_{\ell,k+1}^{(f)}(y) &= \mathsf{na} + \left\lfloor \frac{\bar{R}_{\ell,k}^{(f)}}{\Theta^{(f)}} \right\rfloor \times \Pi^{(f)} \\ &\quad + \mu_{\geq 0}(\mathrm{ps}) \cdot [\Pi^{(f)} - \Theta^{(f)} + \mathrm{ps}] \end{aligned} \tag{7.8}$$

where $\mathsf{na} \stackrel{\text{def}}{=} \Pi^{(f)} + \Delta^{(f)} - 2\Theta^{(f)}$ and $\mathrm{ps} \stackrel{\text{def}}{=} \bar{R}_{\ell,k}^{(f)}(y) - \left\lfloor \frac{\bar{R}_{\ell,k}^{(f)}(y)}{\Theta^{(f)}} \right\rfloor \times \Theta^{(f)}$. Equation 7.8 looks for the start time of y-th job. If the job is not fully non-preemptive, Equation 7.9 (Line 1) calculates $R_\ell^{(f)}(y)$ using the start of the next job.

$$R_\ell^{(f)}(y) = \begin{cases} R_\ell^{(f)}(y+1) - (y-1) \times p_\ell^{(f)}, \text{ if } e_\ell^{(f)} > \mathrm{ne}_\ell^{(f)} \\ R_\ell^{(f)}(y) - (y-1) \times p_\ell^{(f)} + e_\ell^{(f)}, \text{ otherwise.} \end{cases} \tag{7.9}$$

Theorem 4 implies that we may consider the synchronous arrival sequence where each task in the new mode produces a job at time $t_k + \delta_{ij}$ and subsequent jobs as soon as legally permitted by the task specification. The analysis of Equations 7.8 and 7.9 are obtained by extending the response time analysis of Davis et al. [29] to account for the non-availability of the periodic resources and some preemptive execution.

With BI_3, unchanged tasks may start during the transition. So, we may not be able to align all jobs at the start of the busy interval especially all new-mode added tasks. The following theorem addresses unchanged-task schedulability in BI_3.

Theorem 5 *For a* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$*, a task* τ_f *(*$\in \tau^{(ij)}$*) will not miss a deadline due to* BI_3 *if*

$$\begin{aligned} &\forall s, \phi \in \mathbb{R} : (0 \le s < \delta_{ij}) \wedge (\phi > 0) \wedge (s + \phi \ge d_f) :: \\ &\exists t \in [\max(s, s+\phi-d_f), s+\phi] :: \\ &\sum_{\tau_c^{(ij)} \in \mathsf{hp}(\tau^{(ij)}, \tau_f)} \mathsf{rbf}\left(\tau_c^{(ij)}, t\right) + \sum_{\tau_c^{(j)} \in \mathsf{hp}(\tau^{(j)} \setminus \tau^{(ij)}, \tau_f)} \mathsf{rbf}\left(\tau_c^{(j)}, (t-s)_+\right) \\ &+ B(\tau^{(ij)}, \tau_f) \le \beta^{i,j}_{\mathsf{post}}\left(s, (t-s)_+\right) \end{aligned} \tag{7.10}$$

Proof: Assume there is a deadline miss due to BI_3. Consider the scenario in Figure 7.3:

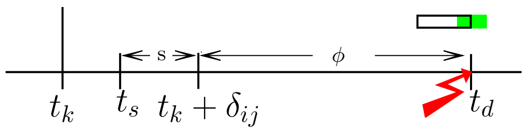

Figure 7.3: Deadline miss event for BI_3.

As the processor is continuously busy with higher-priority tasks, the following must be true by Equation 7.6:

$$\forall t_i \in [t_s, t_d] : \; b + \eta(\tau^{(j)}, t_s, t_i) > \beta(t_s, t_i) \tag{7.11}$$

The total request $\eta(\tau^{(j)}, t_s, t_i)$ is coming from higher-priority unchanged task and new-mode added tasks. In order to achieve known upper bound, we apply job reorganization (similar to the critical instant theorem for unimode scheduling) without reducing the total workload. We shift all tasks to start as soon as legally allowed in the interval to achieve the maximum workload. This shift will only increase the requests in the interval.

1. $\mathsf{hp}\left(\tau^{(ij)}, \tau_f\right)$: Move the first job of all tasks to t_s and allow all tasks to generate jobs as soon as possible. This is valid as the processor is continuously busy and we are increasing the workload.

2. $\mathsf{hp}\left(\tau^{(j)} \setminus \tau^{(ij)}, \tau_f\right)$: Move the first job of all task to $t_k + \delta_{ij}$ and allow all tasks to generate jobs as soon as legally allowed.

For 1), the first rbf of Equation 7.10 is an upper bound and for 2) the second rbf is an upper bound. Furthermore, $B(\tau^{(ij)}, \tau_f)$ is an upper bound on b and $\beta^{i,j}_{\mathsf{post}}(s,t)$ is a lower bound on $\beta(t_s, t_i)$. So, the preceding transformation will violate existential quantifier at Line 2 for Equation 7.10. ■

Corollary 8 *For a* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$*, a task* τ_f *(*$\in \tau^{(j)} \setminus \tau^{(ij)}$*) will not miss a deadline due to* BI_3 *if*

$$\begin{aligned}
&\forall s, \phi \in \mathbb{R} : (0 \le s < \delta_{ij}) \wedge (\phi > 0) \wedge (d_f \le \phi) :: \\
&\quad \exists t \in [s + \phi - d_f, s + \phi] :: \\
&\quad\quad \sum_{\tau_c^{(ij)} \in \mathsf{hp}(\tau^{(ij)}, \tau_f)} \mathsf{rbf}\left(\tau_c^{(ij)}, t\right) + \sum_{\tau_c^{(j)} \in \mathsf{hp}(\tau^{(j)}, \tau_f)} \mathsf{rbf}\left(\tau_c^{(j)}, t - s\right) \\
&\quad\quad + B(\tau^{(ij)}, \tau_f) \le \beta^{i,j}_{\mathsf{post}}(s, (t-s)_+)
\end{aligned} \tag{7.12}$$

New-mode added tasks (i.e., $\tau^{(j)} \setminus \tau^{(ij)}$) will not miss a deadline before $t_k + \delta_{ij} + d_f$; therefore, Corollary 8 differs in the bound of ϕ and t from Theorem 5.

For intra-mode schedulability, we did not have to consider remaining execution requests from the old mode. We denote the execution requests that originate at some old mode but not finished at an MCR as *carry-in*. To prevent a scenario described by the intervals BI_4 and BI_5, we must account for carry-in with the schedulability analysis. The subsequent section on inter-mode schedulability will require reasoning about carry-in execution.

7.3.2 Inter-Mode Schedulability

To determine the schedulability with less pessimism, we must carefully quantify the workload that may *originate* at the old mode but not completed. Over-estimation of carry-in results in pessimism in the schedulability analysis, whereas the under-estimation may generate incorrect results. For obtaining a better insight, we first characterize carry-in for a **concrete sequence** of mode change requests $\mathsf{mcr}_0, \mathsf{mcr}_1, \mathsf{mcr}_2, \ldots$ where timestamps for each MCR are known a priori. Next, we extend the results for the upper bound on carry-in for any sequence of MCRs. We denote such sequence as (**non-concrete**) MCRs. We adopted a similar technique used in the previous chapter (i.e., Fisher and Ahmed [35]) to calculate the carry-in for EDF.

Maximum Carry-In for Concrete MCRs

Definition 14 (carry-in) *For* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$, *the* carry-in $\mathsf{ci}(\mathsf{mcr}_k, \tau_f)$ *is the maximum remaining execution at* $t_k + \delta_{ij}$ *from higher-priority finished tasks* $\mathsf{hp}\left(\tau^{(i)}, \tau_f\right)$ *with arrival before* t_k *and from unchanged tasks (i.e.,* $\mathsf{hp}\left(\tau^{(ij)}, \tau_f\right)$*) with arrival before* $t_k + \delta_{ij}$.

We quantify carry-in $\mathsf{ci}(\mathsf{mcr}_k, \tau_f)$ in terms of request-bound-function. So, we define mode-change-rbf that calculates rbf with respect to an MCR. The following paragraphs derive bounds using *mode-change-rbfs* for different types of tasks.

Definition 15 (mode-change-rbf) *Given* $\mathsf{mcr}_0, \mathsf{mcr}_1, \ldots$*, for any* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$*, and* $x, \phi \in \mathbb{R}_{\geq 0}$*, the* $\mathsf{mcrbf}_k(\tau_\ell^{(i)}, x, \phi)$ *is the maximum execution request of jobs of* $\tau_\ell^{(i)}$ *in the interval* $[t_k - x, t_k + \phi]$.

Lemma 19 *Given* $\mathsf{mcr}_0, \mathsf{mcr}_1, \mathsf{mcr}_2, \ldots$*, for any* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$*,* $\tau_\ell^{(i)} \in \vartheta^{(ij)}$*, and* $x, \phi \geq 0$*,* $\mathsf{mcrbf}_k(\tau_\ell^{(i)}, x, \phi)$ *is maximized by job arrival sequence where the last job of* $\tau_\ell^{(i)}$ *arrives just before* t_k *and previous jobs arrive as late as legally allowed.*

The proof of this lemma is left to the appendix since they are similar to the ones found in [35]. The following corollary computes the upper bound by counting jobs in that sequence:

Corollary 9 *For* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$*,* $\tau_\ell^{(i)} \in \vartheta^{(ij)}$*, and* $x, \phi \geq 0$*,*

$$\mathsf{mcrbf}_k\left(\tau_\ell^{(i)}, x, \phi\right) \leq \left(\left\lfloor \frac{x}{p_\ell^{(i)}} \right\rfloor + 1\right) \times e_\ell^{(i)}. \tag{7.13}$$

Lemmas 20 and 21 generalize the preceding technique to calculate an upper bound on the execution requests for aborted and unchanged tasks. The proof for Lemma 20 can be found in the appendix, whereas the proof for Lemma 21 is omitted since unchanged tasks behave identical to standard unimode tasks in this scenario.

Lemma 20 *For* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$*,* $\tau_\ell^{(i)} \in \alpha^{(ij)}$*,* $x, \phi \geq 0$*, where* $s \overset{\mathit{def}}{=} x - \left\lfloor \frac{x}{p_\ell^{(i)}} \right\rfloor p_\ell^{(i)}$*,*

$$\mathsf{mcrbf}_k\left(\tau_\ell^{(i)}, x, \phi\right) \leq \left\lfloor \frac{x}{p_\ell^{(i)}} \right\rfloor \cdot e_\ell^{(i)} + \mu_{\geq 0}(s) \cdot \min\left(s, e_\ell^{(i)}\right). \tag{7.14}$$

Lemma 21 *For* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$*,* $x, \phi \geq 0$*,*

$$\mathsf{mcrbf}_k\left(\tau_\ell^{(ij)}, x, \phi\right) = \mathsf{rbf}(\tau_\ell^{(ij)}, x + \phi). \tag{7.15}$$

The upper bounds on mcrbf_k are independent of the actual timestamps of a mcr_k, so we may interchangeably denote this function simply by mcrbf (i.e., suffix removed). Observe that mcrbf does not consider any of the preceding mode-change requests. So, we define $\Psi(\mathsf{mcr}_k, \phi, \tau_f)$ that accounts for carry-in and quantifies the maximum remaining execution of higher-priority finished tasks $\mathsf{hp}\left(\vartheta^{(ij)}, \tau_f\right)$ that arrive prior to t_k and the maximum remaining execution of unchanged tasks $\mathsf{hp}\left(\tau^{(ij)}, \tau_f\right)$ that arrive before $t_k + \phi$. Using Lemmas 19, 20, 21 and Corollary 9, we may obtain an upper bound on Ψ and ci as follows:

Lemma 22 *Consider* $\phi \geq 0$ *and successive mode change requests* $\mathsf{mcr}_0, \mathsf{mcr}_1, \ldots$*; for any* $\mathsf{mcr}_{k-1} = \left(M^{(h)},\ M^{(i)},\ t_{k-1}\right)$ *and* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$*, if there are no deadline misses prior to* t_{k-1}*, the upper-bound on carry-in rbf at* ϕ *after* mcr_k *is as follows:*

$$\Psi(\mathsf{mcr}_k, \phi, \tau_f) \quad \leq \sup_{0 \leq x \leq t_k - t_{k-1} - \delta_{hi}} \begin{cases} \displaystyle\sum_{\tau_\ell^{(i)} \in \mathsf{hp}(\tau^{(i)}, \tau_f)} \mathsf{mcrbf}\left(\tau_\ell^{(i)}, x, \phi\right) \\ +\mathit{IF}\left(x \geq (t_k - t_{k-1} - \delta_{hi}),\right. \\ \quad \left.\mathsf{ci}(\mathsf{mcr}_{k-1}, \tau_f), B(\tau^{(h)}, \tau_f)\right) \\ \quad -\beta^i_{\mathsf{prior}}(x) \end{cases} \tag{7.16}$$

such that $\mathsf{ci}(\mathsf{mcr}_{k-1}, \tau_f)$ *is upper bounded by the minimum of*

$$\sum_{\tau_\ell^{(hi)} \in \mathsf{hp}\left(\tau^{(hi)}, \tau_f\right)} e_\ell^{(hi)} + \left(\begin{array}{l} \sum\limits_{\tau_\ell^{(h)} \in \mathsf{hp}\left(\vartheta^{(hi)}, \tau_f\right)} e_\ell^{(h)} \\ + \sum\limits_{\tau_\ell^{(hi)} \in \mathsf{hp}\left(\tau^{(hi)}, \tau_f\right)} \left\lfloor \frac{\delta_{hi}}{p_\ell^{(hi)}} \right\rfloor e_\ell^{(hi)} \\ + B(\tau^{(h)}, \tau_f) - \beta^{i,j}_{\mathsf{trans}}(\delta_{hi}) \end{array} \right)_+ \tag{7.17}$$

and

$$\max \left(\Psi(\mathsf{mcr}_{k-1}, \delta_{hi}, \tau_f) - \beta^{h,i}_{\mathsf{trans}}(\delta_{hi}) \ , \sum_{\tau_\ell^{(h)} \in \mathsf{hp}\left(\tau^{(hi)}, \tau_f\right)} e_\ell^{(h)} \right). \tag{7.18}$$

For improving readability, the detailed proof is sent to the appendix. For intuition, Equation 7.16 calculates the maximum carrying considering all possible busy intervals around t_k. Equation 7.18 gives the carry-in that the mode $M^{(i)}$ may start with after the immediate previous MCR for which Equation 7.17 is the upper bound.

Max Carry-In for Non-Concrete MCRs

Calculation of carry-in developed so far requires the full knowledge of all past MCRs which may not be practicable. In case of non-concrete MCRs, the carry-in can be upper-bounded by applying an iterative calculation. We use the notations $\mathsf{ci}^{i,j}$ to denote the maximum carry-in execution and $\Psi^{i,j}$ as carry-in request-bound function for a mode change from $M^{(i)}$ to any other mode $M^{(j)}$.

Definition 16 (max-carry-in) *The* $\mathsf{ci}^{i,j}(\tau_f)$ *for a mode change from* $M^{(i)}$ *to any other mode* $M^{(j)}$ *is an upper bound at time* $t_k + \delta_{ij}$ *on the remaining execution, with priority as* τ_f*, from finished tasks (that arrive prior to* t_k*) and unchanged tasks (that arrive prior to* $t_k + \delta_{ij}$*).*

Definition 17 (max-mode-change-rbf) *Given a mode* $M^{(i)}$ *starting with carry-in* ζ*, for a mode change from* $M^{(i)}$ *to* $M^{(j)}$ *at time* t_k*, the* $\Psi^{i,j}(\zeta, \phi, \tau_f)$ *is the maximum higher-priority remaining execution (over any legal sequence of MCRs prior to* t_k*) of* $\vartheta^{(ij)}$ *(that arrive prior to* t_k*) and of* $\tau^{(ij)}$ *(that arrive prior to* $t_k + \delta_{ij}$*).*

The preceding function can be calculated as the $\Psi^{i,j}(\zeta, \phi, \tau_f) \overset{\text{def}}{=} \sup_{x>0} \Psi_x^{i,j}(\zeta, \phi, \tau_f)$. An upper bound on this function immediately follows from Equation 7.16:

$$\Psi_x^{i,j}(\zeta, \phi, \tau_f) \leq \begin{cases} \sum_{\tau_\ell^{(i)} \in \mathsf{hp}(\tau^{(i)}, \tau_f)} \mathsf{mcrbf}\left(\tau_\ell^{(i)}, x, \phi\right) \\ +\text{IF}\left(x \geq (t_k - t_{k-1} - \delta_{\max}^i), \zeta, B(\tau^{(i)}, \tau_f)\right) \\ -\beta_{\mathsf{prior}}^i(x) \end{cases} \tag{7.19}$$

where $\delta_{\max}^i \overset{\text{def}}{=} \max_h \delta_{hi}$. As in Equation 7.19, the calculation of carry-in rbf at $M^{(j)}$ is dependent on the carry-in ζ that the previous mode $M^{(i)}$ may receive as carry-in. So, unlike a concrete sequence, the exact calculation of the carry-in at $M^{(j)}$ is difficult. So for the upper bound, we evaluate the sequence $\mathsf{ci}_0^{i,j}(\tau_f)$, $\mathsf{ci}_1^{i,j}(\tau_f)$, $\mathsf{ci}_2^{i,j}(\tau_f)$, … for all $i, j (i \neq j) \in \{1, \ldots, q\}$ where (for any $\eta \in \mathbb{N}$),

$$\mathsf{ci}_\eta^{i,j}(\tau_f) \overset{\text{def}}{=} \begin{cases} 0, & \text{if } \eta = 0, \\ \min\left(E_{ij}(\tau_f), \max\limits_{\substack{h=1,\ldots,q \\ h \neq i}} F_{ij}(\{\mathsf{ci}_{\eta-1}^{h,i}(\tau_f)\}, \tau_f)\right), & \text{if } \eta > 0. \end{cases} \tag{7.20}$$

where $E_{ij}(\tau_f)$ equals

$$\sum_{\tau_\ell^{(ij)} \in \mathsf{hp}(\tau^{(ij)}, \tau_f)} e_\ell^{(ij)} + \left(\begin{array}{l} \sum_{\tau_\ell^{(i)} \in \mathsf{hp}(\vartheta^{(ij)}, \tau_f)} e_\ell^{(i)} \\ + \sum_{\tau_\ell^{(ij)} \in \mathsf{hp}(\tau^{(ij)}, \tau_f)} \left\lfloor \frac{\delta_{ij}}{p_\ell^{(ij)}} \right\rfloor e_\ell^{(ij)} \\ \quad +B(\tau^{(i)}, \tau_f) - \beta_{\mathsf{trans}}^{i,j}(\delta_{ij}) \end{array} \right)_+ , \tag{7.21}$$

and $F_{ij}(\zeta, \tau_f)$ equals

$$\max\left(\Psi^{i,j}(\zeta, \delta_{ij}, \tau_f) - \beta_{\mathsf{trans}}^{i,j}(\delta_{ij}) \; , \sum_{\tau_\ell^{(i)} \in \mathsf{hp}(\tau^{(ij)}, \tau_f)} e_\ell^{(i)} \right). \tag{7.22}$$

For any pair of modes $M^{(i)}$ and $M^{(j)}$, we need to show that the carry-in after ℓ-th MCR is always bounded from above by $\mathsf{ci}_{\ell}^{i,j}(\tau_f)$.

Lemma 23 *For a mode-change request* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$ *following a sequence of* k *mode change requests,* $\mathsf{mcr}_0, \mathsf{mcr}_1 \ldots \mathsf{mcr}_{k-1}$*, the carry-in with priority as* τ_f *is at most* $\mathsf{ci}_{k}^{i,j}(\tau_f)$ *if there is no deadline miss prior to* t_k*.*

A formal proof can be found in the appendix. Intuitively, we show that the sequence $\mathsf{ci}_{0}^{i,j}(\tau_f)$, $\mathsf{ci}_{1}^{i,j}(\tau_f)$, $\mathsf{ci}_{2}^{i,j}(\tau_f)$, ... for all $i, j (i \neq j) \in \{1, \ldots, q\}$ is monotonically non-decreasing and the maximum value for each $\mathsf{ci}_{\ell}^{i,j}(\tau_f)$ is bounded from above by Equation 7.21. The value of $\mathsf{ci}^{i,j}(\tau_f)$ can be obtained from the convergence of this sequence. The convergence occurs at the smallest $g \in \mathbb{N}$ such that $\forall i, j (i \neq j) \in \{1, \ldots, q\}$, $c_g(M^{(i)}, M^{(j)}, \tau_f) = c_{g-1}(M^{(i)}, M^{(j)}, \tau_f)$. Using carry-in, the following sections establish conditions for a missed deadline. We utilize $\zeta_j(\tau_f)$ which equals to $\max\limits_{i=1,\ldots,q \wedge i \neq j} \{\mathsf{ci}^{i,j}(\tau_f)\}$.

The largest busy-interval for BI_4 and BI_5 partially depends on carry-in. While calculating the longest busy-interval, Equation 7.7 does not consider the carry-in. The following corollary is an immediate extension:

Corollary 10 *For a MM system, the busy-interval for a task* $\tau_{\ell}^{(j)}$ *after a mode-change request depends on both request and supply. The largest busy-interval* $L_{\ell}^{(j)}$ *is a recurrence as follows:*

$$\begin{aligned}
L_{\ell,0}^{(j)} &\overset{def}{=} \zeta_j(\tau_{\ell}^{(j)}) + B(\tau_{\ell}^{(j)}, \tau^{(j)}) + e_{\ell}^{(j)} \\
\bar{L}_{\ell,m}^{(j)} &\overset{def}{=} L_{\ell,0}^{(j)} + \sum_{\tau_k^{(j)} \in \mathsf{hp}\left(\tau^{(j)}, \tau_{\ell}^{(j)}\right)} \left\lceil \frac{L_{\ell,m}^{(j)}}{p_k^{(j)}} \right\rceil \times e_k^{(j)} \\
L_{\ell,m+1}^{(j)} &\overset{def}{=} \mathsf{na} + \left\lfloor \frac{\bar{L}_{\ell,m}^{(j)}}{\Theta^{(j)}} \right\rfloor \times \Pi^{(j)} + \mu_{\geq 0}(s) \cdot [\Pi^{(j)} - \Theta^{(j)} + s]
\end{aligned} \tag{7.23}$$

where $\mathsf{na} \overset{def}{=} \Pi^{(j)} + \Delta^{(j)} - 2\Theta^{(j)}$ *and* $s \overset{def}{=} \bar{L}_{\ell,m+1}^{(j)} - \left\lfloor \frac{\bar{L}_{\ell,m+1}^{(j)}}{\Theta^{(j)}} \right\rfloor \times \Theta^{(j)}$.

Using the largest busy-interval length, we develop analysis for unchanged, finished, and new-mode added tasks. The function $\Psi_x^{i,j}(\zeta, \phi, \tau_f)$ essentially considers rbf for all higher-priorities in the interval length $x+\phi$, so it will over-estimate for the task τ_f. We consider demand analysis for only τ_f since its last job has deadline within the busy-interval and thus all of its jobs in the busy interval have both arrivals and deadline in the interval. So, to reduce pessimism, we consider a mix of demand-bound and request-bound analysis in the function $\hat{\Psi}$ as follows:

$$\hat{\Psi}_x^{i,j}(\zeta,\phi,\tau_f) \leq \begin{cases} \sum_{\tau_\ell^{(i)} \in \mathsf{hp}\left(\tau^{(i)} \setminus \tau_f, \tau_f\right)} \mathsf{mcrbf}\left(\tau_\ell^{(i)}, x, \phi\right) \\ + \left(\left\lfloor \frac{(x+\phi-d_f)_+}{p_f} \right\rfloor + 1\right) \times e_f \\ + \text{IF}\left(x \geq (t_k - t_{k-1} - \delta_{\max}^i), \zeta, B(\tau^{(i)}, \tau_f)\right) \\ - \beta_{\mathsf{prior}}^i(x) \end{cases} \tag{7.24}$$

The second line of Equation 7.24 (i.e., $(\lfloor (x+\phi-d_f)_+/p_f \rfloor + 1) \times e_f$) considers all jobs τ_f that have arrival and deadline in $x+\phi$ length interval. Our supply analysis requires deduction of the transition period supply, for which, we utilize the following function:

$$\text{RBF}_o(\zeta, x, \phi, \tau_f) = \begin{cases} \hat{\Psi}_x^{i,j}(\zeta,\phi,\tau_f) - \beta_{\mathsf{trans}}^{i,j}(\delta_{ij}), & \text{if } \hat{\Psi}_x^{i,j}(\zeta,\delta_{ij},\tau_f) \geq \beta_{\mathsf{trans}}^{i,j}(\delta_{ij}) \\ \sum_{\tau_c^{(ij)} \in \mathsf{hp}\left(\tau^{(ij)}, \tau_f\right)} \mathsf{rbf}\left(\tau_c^{(ij)}, (\phi - \delta_{ij})_+\right), & \text{otherwise.} \end{cases} \tag{7.25}$$

The above equation considers jobs that could be generated after $t_k + \delta_{ij}$ if supply during transition is greater than carry-in jobs. As the RBF$_o$ calculates request bound of tasks that may belong to the old mode, we take the suffix 'o' as in 'old'. Theorems 6 and 7 utilize RBF$_o$ function to calculate contribution from old-mode finished tasks $\vartheta^{(ij)}$ and unchanged tasks $\tau^{(ij)}$ while checking schedulability of $M^{(j)}$ after a mode change from $M^{(i)}$.

Theorem 6 *For a* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$*, a finished task* $\tau_\ell^{(i)}$ *(*$\in \vartheta^{(ij)}$*) is schedulable if*

$$\begin{aligned}
&\forall \phi \in [0, d_\ell^{(i)}], \forall x : d_\ell^{(i)} \leq x + \phi \\
&\quad \exists t \in [\max(x, x + \phi - d_\ell^{(i)}), x + \phi], \mathit{mt} \overset{\mathit{def}}{=} (t - x - \delta_{ij})_+ \\
&\quad \mathit{RBF}_o(\mathsf{ci}^{i,j}(\tau_\ell^{(i)}), x, t - x, \tau_\ell^{(i)}) \\
&\qquad + \sum_{\tau_c^{(j)} \in \mathsf{hp}\left(\tau^{(j)} \setminus \tau^{(ij)}, \tau_\ell^{(i)}\right)} \mathsf{rbf}\left(\tau_c^{(j)}, \mathit{mt}\right) \leq \beta_{\mathsf{post}}^{i,j}(0, \mathit{mt})
\end{aligned} \tag{7.26}$$

Proof: Theorem 6 checks whether a single job that may start at $t_k + \phi - d_\ell^{(i)}$ would receive enough execution supply to complete its requirement before $t_k + \phi$. The variable t in the above equation denotes the length of intervals that start with $t_k - x$ and ends before $t_k + \phi$.

We sketch a proof of this theorem for a busy interval that originates (let us say at $t_k - x$) while executing in a previous mode and finishes (let us say at $t_k + \phi$) in $M^{(j)}$ where the length of the busy-interval is $x + \phi$. As there could be a single job of $\tau_\ell^{(i)}$ in $M^{(j)}$, we consider all possible scenarios related to the execution of a job of $\tau_\ell^{(i)}$ that arrives in $M^{(i)}$, but missed its deadline in the next mode $M^{(j)}$. The following is a case-based study:

1. §$t_k - (t_{k-1} + \delta_{hi}) > x$**.** The first case assumes that the origin $t_k - x$ is in immediate previous mode $M^{(i)}$. Now we reorganize the jobs in $[t_k - x, t_k + \phi]$ without reducing the workload to achieve a known sequence for the upper bound on requests. We align the jobs of $\tau_\ell^{(i)}$ with respect to $t_k + \phi$; that is all the previous jobs before the one that missed the deadline arrive as late as legally allowed. The first job of each $\tau_k^{(ij)} \in \mathsf{hp}\left(\tau^{(ij)}, \tau_\ell^{(i)}\right)$, could be moved to $t_k - x$ and all the subsequent jobs could arrive as soon as legally possible. The higher-priority aborted jobs and the finished task of $\tau^{(i)}$ could be reorganized applying the sequence of Lemmas 19 and 20 to obtain the largest interference while in $M^{(i)}$. The exact upper bound on the request from higher-priority tasks is quantified in RBF$_o$ (i.e., the first line of Equation 7.24). There could be blocking from lower-priority tasks which is addressed at Line 3 of Equation 7.24. Now we may need to consider two additional cases based on ϕ:

(a) $\S\phi > \delta_{ij}$**.** There could be jobs from new-mode tasks. For this case, the LHS of Line 4 of Equation 7.26 quantifies the upper bound of jobs considering the fact that all the new-mode added tasks may generate at $t_k + \delta_{ij}$. The RHS of Line 4 quantifies the minimum supply in $M^{(j)}$.

(b) $\S\phi \leq \delta_{ij}$**.** There will not be any job requests and execution supply in $M^{(j)}$ as the busy-interval terminates before $t_k + \delta_{ij}$. Both sides of the inequality at Line 4, Equation 7.26 are evaluated to zero for this case.

For the above two cases, the request will be evaluated greater than supply for a missed deadline. So, if Equation 7.26 holds, there is no way a job from finished task would miss a deadline in the new-mode after a mode-change request.

2. $\S t_k - (t_{k-1} + \delta_{hi}) \leq x$**.** This second case considers that the origin of the busy-interval could be any previously executing mode before $M^{(i)}$. Without loss of generality, we further assume that the system is in mode $M^{(i)}$ for x_1 amount of time and for $x - x_1$ in all previous modes. In addition to the maximum startup carry-in $\mathsf{ci}^{i,j}(\tau_\ell^{(i)})$, for the interval $[t_k - x_1, t_k + \phi]$, there could be at most $\lfloor (x_1 + \phi)/p_k^{(ij)} \rfloor e_k^{(ij)} + \min(e_\ell^{(ij)}, x_1 + \phi - \lfloor (x_1 + \phi)/p_k^{(ij)} \rfloor p_\ell^{(ij)})$, but we considered $(1 + \lfloor (x_1 + \phi)/p_k^{(ij)} \rfloor) e_k^{(ij)}$ as a safe upper-bound for all possible x_1 values.

Considering all the above cases, we may claim that if Equation 7.26 holds, no job will miss a deadline. ■

Example 1 To illustrate Theorem 6, we utilize the multimode system in Figure 7.1. Assume there are no transition periods between modes (i.e., $\delta_{12} = 0$ and $\delta_{21} = 0$) and $N = 2$ for all modes. Now consider any two MCRs $\mathsf{mcr}_{k-1} = (M^{(1)}, M^{(2)}, t_{k-1})$ and $\mathsf{mcr}_k = (M^{(2)}, M^{(1)}, t_k)$ separated by 20 units (e.g., $t_{k-1} = 20$ and $t_k = 40$). We consider all $t_k + \phi$ where $\phi \in [0, 10]$ (Line 1 of Equation 7.26) as the deadline for the last job of $\tau_3^{(2)}$ in the new-mode $M^{(1)}$. Using the second universal quantifier (Line 1), we evaluate all valid busy intervals of length x (e.g., $\phi = 1$ requires all $x \geq 9$). The existential quantifier for t (Line 2) checks whether the last job receives sufficient supply over any possible busy interval. Finally, RBF$_o$ (Line 3) performs request/demand based analysis for each t. For $\phi = 1$ and $x = 20$, t must be in $[20, 21]$. For $t = 20$, the last two terms of Equation 7.26 are zero as $mt = 0$, so the existential quantifier becomes true as $6 + 6 + 2 + \zeta < \beta^{2}_{\mathsf{prior}}(20) + \beta^{2,1}_{\mathsf{trans}}(0) + \beta^{2,1}_{\mathsf{post}}(0, 0)$ where the carry-in ζ for $\tau_3^{(2)}$ in $M^{(2)}$ is 3, and $\beta^{2}_{\mathsf{prior}}(20) = 20$. ■

For the Theorem 6, we consider a single job after an MCR as there could be only one job possible from a finished task after a mode change. For unchanged tasks, we apply a similar approach and consider each $t_k + \phi$ as a possible candidate for a deadline miss. We start looking for a single deadline miss from the end of a busy-interval. This is sufficient as all ϕ are considered for a deadline misses.

Theorem 7 *For a* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$*, an unchanged task* $\tau_\ell^{(ij)}$ *(*$\in \tau^{(ij)}$*) is schedulable if*

$$\begin{aligned}
&\forall 0 < \phi \le L_\ell^{(ij)}, \forall x : d_\ell^{(ij)} \le x + \phi \\
&\exists t \in [\max(x, x + \phi - d_\ell^{(ij)}), x + \phi], mt \overset{def}{=} (t - x - \delta_{ij})_+ \\
&\quad RBF_o(\mathsf{ci}^{i,j}(\tau_\ell^{(ij)}), x, t - x, \tau_\ell^{(ij)}) \\
&\qquad + \sum_{\tau_c^{(j)} \in \mathsf{hp}\left(\tau^{(j)} \setminus \tau^{(ij)}, \tau_\ell^{(j)}\right)} \mathsf{rbf}\left(\tau_c^{(j)}, mt\right) \le \beta^{i,j}_{\mathsf{post}}(\delta_{ij}, mt).
\end{aligned} \tag{7.27}$$

Proof: The proof is very similar to the Theorem 6; therefore, we provide here a proof sketch for a busy interval that originates (let say at $t_k - x$) while executing in some previous mode and ends at (let say $t_k + \phi$) in $M^{(j)}$. We consider the last job of an unchanged task $\tau_\ell^{(ij)}$ for request/supply analysis in a $\phi + x$ length interval. We reorganize jobs in $[t_k - x, t_k + \phi]$ to a known sequence for the upper bound on requests. We align the jobs of $\tau_\ell^{(ij)}$ with respect to $t_k + \phi$; that is, all the previous jobs before the one that missed the deadline arrive as late as legally allowed. We consider two cases as follows:

1. **§$t_k - (t_{k-1} + \delta_{hi}) > x$.** The first case assumes that the origin is in immediate previous mode $M^{(i)}$. The first job of each $\tau_k^{(ij)} \in \mathsf{hp}\left(\tau^{(ij)}, \tau_\ell^{(ij)}\right)$, where $k \ne \ell$, could be moved

to $t_k - x$ and all the subsequent jobs could arrive as soon as possible. The higher-priority aborted jobs and the finished task of $\tau^{(i)}$ could be reorganized following the sequence of Lemma 19 to obtain the largest interference while in $M^{(i)}$. The exact upper bound on the execution is quantified in RBF$_o$ (the first line of Equation 7.24).

2. **§$t_k - (t_{k-1} + \delta_{hi}) \leq x$.** This second case considers the origin could be any executing previous mode before $M^{(i)}$. Without loss of generality, we further assume that the system is in mode $M^{(i)}$ for x_1 amount of time, and $x - x_1$ in all previous modes. In the interval $[t_k - x_1, t_k + \phi]$, there could be at most $\lfloor (x_1 + \phi)/p_k^{(ij)} \rfloor e_k^{(ij)} + \min(e_\ell^{(ij)}, x_1 + \phi - \lfloor (x_1 + \phi)/p_k^{(ij)} \rfloor p_\ell^{(ij)})$, but we quantified $(1 + \lfloor (x_1 + \phi)/p_k^{(ij)} \rfloor) e_k^{(ij)}$ as a safe upper-bound.

Considering all the above cases, we may claim that if Equation 7.27 holds, then no way a job can miss a deadline. ■

Using the same technique, the schedulability for new-mode added tasks with BI_4 and BI_5 can be achieved as follows, using the upper-bound on busy-interval $L_\ell^{(j)}$ from Equation 7.23.

Corollary 11 *For a* $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$, *a new-mode task* $\tau_\ell^{(j)} \in \tau^{(j)}$ *is schedulable if*

$$\begin{aligned}
&\forall \phi \in [\delta_{ij} + d_\ell^{(j)}, L_\ell^{(j)}], \forall x > 0 \\
&\exists t \in [x + \phi - d_\ell^{(j)}, x + \phi], mt \overset{def}{=} (t - x - \delta_{ij})_+ \\
&\quad RBF_o(\mathsf{ci}^{i,j}(\tau_\ell^{(j)}), x, t - x, \tau_\ell^{(j)}) \\
&\quad + \sum_{\tau_c^{(j)} \in \mathsf{hp}\left(\tau^{(j)} \setminus \tau^{(ij)}, \tau_\ell^{(j)}\right)} \mathsf{rbf}\left(\tau_c^{(j)}, mt\right) \leq \beta_{\mathsf{post}}^{i,j}(0, mt)\,.
\end{aligned} \tag{7.28}$$

As in the preceding corollary, for new-mode added tasks we do not need to check $\phi < \delta_{ij} + d_\ell^{(j)}$ as they are not allowed to generate jobs before $t_k + \delta_{ij}$.

§**Algorithm & Complexity.** By implementing Theorems 5, 6 and 7, and Corollaries 8 and 11, we develop our algorithm *schedulability using bounded iteration* (SUBI). Theorem 6, 7, and Corollary 11 check all possible x in the old-mode. Using the similar technique by Fisher and Ahmed [35], we include Lemma 26 in the Appendix that limits the number of x to be checked in an old-mode. In addition, the length of busy-intervals are pseudo-polynomial when utilization $u^{(i)}$ is less than $\Theta^{(i)}/\Pi^{(i)}$ of the resource. Therefore, the complexity of SUBI for checking all the conditions are pseudo-polynomial.

§**Hardware non-preemptivity.** We have so far addressed non-preemptive execution from software/application perspective. That is, the analysis assumes that other tasks within the same application may not preempt a task in a non-preemptive region; however, the resource may not be continuously available to the application. Our analysis continues to hold if we assume that the hardware resource cannot be preempted during a non-preemptive region. However, if a subsystem co-executed with other subsystems on the same processor, additional "resource-level" schedulability analysis would be required to deal with such "overruns" (see [14] for details).

7.4 Simulations

We perform two sets of experiments: radar case study and schedulability comparison.

7.4.1 Case Study

To verify the effectiveness of the proposed schedulability, we develop a simple automotive adaptive cruise control (AACC) simulation for demonstrating the practicability of multimode system with non-preemptive tasks. We simulate fixed priority non-preemptive tasks that interact with the external environment through a 77GHz radar. The system estimates the distance of the front vehicle (target), and alerts if the target vehicle is too close. The radar system (implemented using MATLAB Phased Array Toolbox [65]) uses frequency modulated continuous wave (FMCW) technique [46] to measure the target distance and velocity. We performs Doppler estimation [46] to measure distance and velocity of a target moving vehicle. We skip all the details parameter of the radar transmitter/receiver as not relevant to this research, but could be found in Mathworks website [65].

For FMCW radars, the sweep time (from the start of sending waveform through air to the finish time of receiving reflected signal) depends on the required maximum distance. For example, the sweep time is 7.33 μ-seconds [65] for the maximum distance of 200m, whereas it is 1.83 μ-seconds for 50m. We use two radar tasks: short range (SR) and long range (LR) depending on the target distance (i.e., 45m). For moving targets, the accuracy depends also on the number of sweeps used in the measurement. If the target is too close, a higher accuracy is desired; so, we use 16 and 8 sweeps correspondingly with SR and LR radar tasks. Therefore, the worst case non-preemptive executions time are approximately 40 μ-seconds and 80 μ-seconds for SR and LR radar tasks, respectively. In addition, we introduce a control task (C) for tracking target distance and controlling speed (properties listed in Table 7.1). To consume idle cycles, a low criticality preemptive task (LC) with the least priority is always present for better resource utilization.

Table 7.2: Radar Case Study: Mode Resources.

Modes	Capacity (μs)	Period of Repetition (μs)	Condition (m)
$M^{(1)}$	100	100	$D < 20$
$M^{(2)}$	100	100	$D \in (20, 45]$
$M^{(3)}$	90	100	$D > 45$

We develop three modes for the AACC application to exploit SR and LR tasks with properties listed in Tables 7.1 and 7.2. The mode transition occurs based on target distance (D in Table 7.2). For comparison, we also simulate the AACC application using a single mode. Figure 7.4 demonstrates 60 second simulation of AACC vehicle that follows a target approximately $40 \sim 50$m from behind. The top graph is showing the percentage of error (measured distance - actual distance)/(actual distance) in the measurement. The bottom graph is showing reclaimed execution

Tasks	WCET (μs)	Priority	Periods (μs)			
	$e^{(i)}$		$M^{(1)}$	$M^{(2)}$	$M^{(3)}$	Uni Mode
SR	40	1	160	200	0	0
LR	80	2	0	0	160	160
C	10	3	100	100	100	100
LC	∞	4				

Table 7.1: Radar Case Study: Tasks Distribution in Modes.

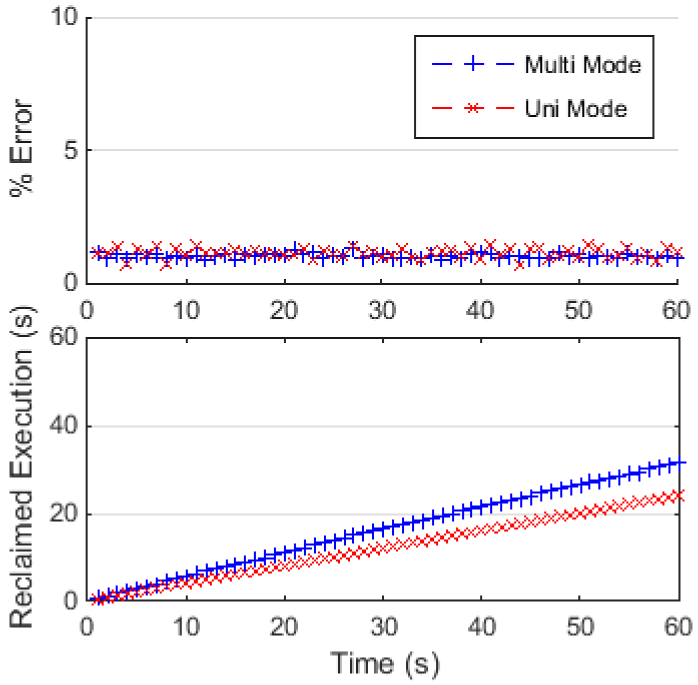

Figure 7.4: Radar Simulation: a) Error in distance measurement for unimode vs multimode, b) Idle time reclamation for unimode vs. multimode.

through low criticality task which is at least 7% more for multimode. Using multimode, we basically exploit the smaller execution requirement of SR task. We perform other simulation with target at most 40m away and observe higher reclamation with improved accuracy. This is due to higher number of sweeps for $M^{(1)}$ and $M^{(2)}$ and less utilization. Prudent choices of modes and parameters [4] may further improve efficacy which is discussed in the next chapter.

7.4.2 Schedulability Comparison

We are not aware of any multimode schedulability analysis that can address non-preemptive execution. To reaffirm the correctness, we perform simulations that compare SUBI with *schedulability analysis using reachability graph* by Phan et al. [63]. We utilize synthetic tasks for comparison with following parameters:

1. The total number of tasks in the system is 8. Task properties and priorities at each mode are described in Table 7.3.

2. During a mode transition, jobs from task τ_1 are considered as aborted jobs. Task τ_5 is unchanged between MCRs involving M_2/M_3.

3. The resource period (Π) and deadline (Δ) are set to 10 for all modes. The offset δ_{ij} is set to Π and $N^{(i)}$ is set equal to 2 for all modes $M^{(i)}$.

In the simulation, we randomly generate a set of capacities ($\Theta^{(i)}$) where the total sum is taken from the range $[1, q\Pi]$. We execute SUBI and SURG for checking schedulability of the subsystem. The graph at the top of the Figure 7.5 presents the percentage of 'YES' responses out of 200 runs on each distinct summation of capacities (i.e., the value on the x axis). The dashed line depicts the results for SURG, and solid line is for SUBI. The graph at the bottom in Figure 7.5 presents the average elapsed time for deciding the schedulability over randomly-generated capacities. For this particular subsystem, Figure 7.5 illustrates that SUBI does as well as SURG and is clearly more computationally efficient.

Tasks	Properties			*Priorities*		
#	$e_\ell^{(i)}$	$d_\ell^{(i)}$	$p_\ell^{(i)}$	M_1	M_2	M_3
1	1	10	10	1	-	1
2	3	30	30	2	2	-
3	4	40	40	-	-	3
4	1	10	10	4	-	-
5	1	20	20	-	3	3
6	3	24	24	-	-	4
7	2	20	20	7	-	-
8	3	30	30	-	1	-

Table 7.3: FP Schedulability Comparison: Tasks Properties.

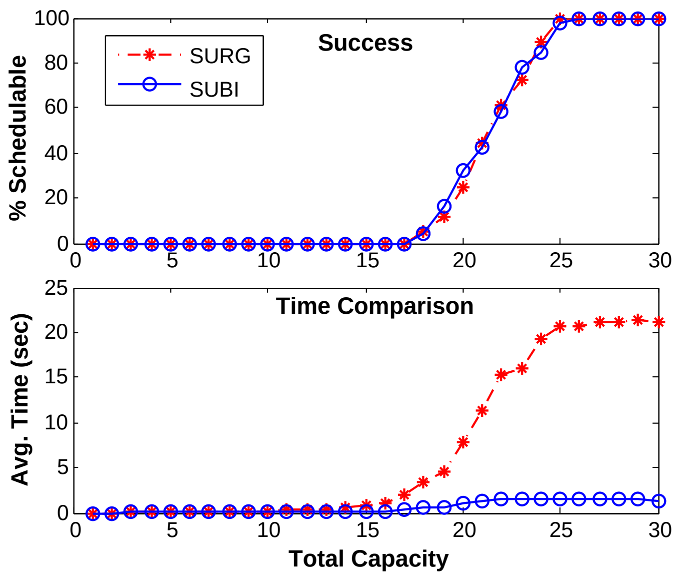

Figure 7.5: FP Schedulability Comparison: SURG vs SUBI.

7.5 Conclusion

In this report, we present an efficient FP-schedulability analysis for multi-modal systems. In addition, our schedulability analysis for multi-modal systems can address non-preemptible execution of a task in a mode. Furthermore, we showed that our analysis can be done in tractable time complexity; therefore, this result may be used to calculate more refined (near optimal) resource parameters by repetitive application of this schedulability with varying hardware parameters which discussed in the next chapter (i.e., Ahmed and Fisher [4]).

Unlike SURG, we do not assume buffers for tasks; so, finished tasks cannot span more than two mode-change requests. In addition, SURG supports hybrid scheduling where a group of tasks (scheduled using EDF) of a mode are assigned to a single buffer, and all buffers are scheduled using FP. Our future research will apply similar iterative approach, as shown in this paper, for multi-modal system with hybrid scheduling to reduce complexity from exponential to polynomial.

Appendix

Proof of Lemma 19 Let J be the set of jobs that arrive according to the sequence described in the lemma with arrivals in $[t_k - x, t_k)$. Assume a different sequence of jobs J' of $\tau^{(i)}$ other than J maximizes the total request over the interval. We will show by induction over the jobs of J' that we may transform J' into J without decreasing the total request. We denote the sequence of jobs (in decreasing order of arrival) as $j_0, j_1, \ldots$ for J and $j'_0, j'_1, \ldots$ for J'. Let j'_k denote the latest arriving job of sequence J' that does have the same arrival-time as j_k in J (i.e, j_i and j'_i arrive at identical times for $i = 0, 1, \ldots, k-1$). As j_k arrives as late as legally possible, it must be that j'_k has an earlier arrival time than j_k. Since j_k has arrival in $[t_k - x, t_k]$, moving arrival time of j'_k to that of j_k will ensure that j'_k is still in the interval and does not violate the minimum interval constraint for $\tau_\ell^{(i)}$. Let's call this new sequence with j'_k moved to the corresponding j_k arrival as J''. It is clear that the total request of J'' does not decrease when compared to J'. By repeated application of this transformation, we may change J' to J without ever decreasing the total request which implies that J also maximizes the total execution requests. ■

Proof of Lemma 20 The mcrbf_k function needs to consider jobs that have only arrival in a given interval. A job from $\alpha^{(ij)}$ aborts at the time of a mode-change request; therefore, a regular rbf for x will overestimate the total request. For any interval of length x, at most $\lfloor x/p_\ell^{(i)} \rfloor$ jobs have their periods completely contained in the x-length interval (i.e., if a job arrives at t and both t and $t + p_\ell^{(i)}$ are in the x-length interval, it is completely-contained). There are at most $\lceil x/p_\ell^{(i)} \rceil$ jobs that can arrive in such an interval (the last one may be only partially contained). Equation 7.14 includes the execution for the completely-contained jobs in the first term.

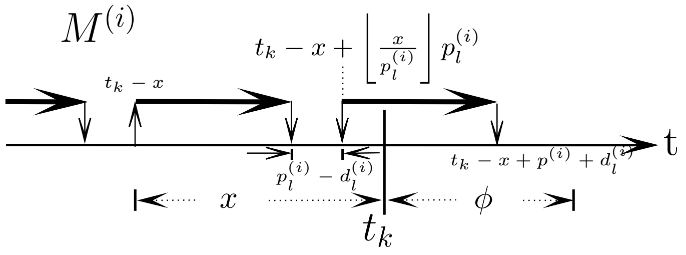
$M^{(i)}$
$t_k - x$
$t_k - x + \left\lfloor \frac{x}{p_l^{(i)}} \right\rfloor p_l^{(i)}$
$p_l^{(i)} - d_l^{(i)}$
$t_k - x + p^{(i)} + d_l^{(i)}$
t
x
ϕ
t_k

Figure 7.6: Worst case arrival for aborted tasks.

The last partially-contained job in the interval $[t_k - x, t_k]$ can arrive at the earliest at $t_k - x + \lfloor x/p_\ell^{(i)} \rfloor p_\ell^{(i)}$. If $x - \lfloor x/p_\ell^{(i)} \rfloor$ is positive, the last job ($\lceil x/p_\ell^{(i)} \rceil$-th) can get partial execution of at most $\min(x - \lfloor x/p_\ell^{(i)} \rfloor p_\ell^{(i)}, e_\ell^{(i)})$ based on the interval length. Equation 7.14 accounts for $\lfloor x/p_\ell^{(i)} \rfloor$ complete jobs along with the last (partially-executed) job which proves our lemma. (A similar observation is made by Pedro et al. [60] in the context of application-only mode changes.) Figure 7.6 illustrates the sequence described in the proof. Downward arrow denotes deadline for each job, while rectangular region depicts execution requirements. ■

Proof of Lemma 22 Let t be the latest time after $t_{k-1} + \delta^{(hi)}$ and the last time prior to t_k during which the processor is executing jobs with priority less than τ_f. Let x be $t_k - t$. Thus, if t equals $t_{k-1} + \delta^{(hi)}$ (i.e., x equals $t_k - t_{k-1} - \delta^{(hi)}$), clearly, the amount of carry-in from $M^{(i)}$ to $M^{(j)}$ is at most the carry-in from mode $M^{(h)}$ (i.e., $\mathsf{ci}(\mathsf{mcr}_{k-1})$), plus the total higher or equal priority requests generated minus the service received over $[t_{k-1} + \delta^{(hi)}, t_k + \phi]$. If t is later than $t_{k-1} + \delta^{(hi)}$ (i.e., $x < t_k - t_{k-1} - \delta^{(hi)}$), then the carry-in from $M^{(i)}$ to $M^{(j)}$ is at most the total requests generated minus the service received over $[t_{k-1} + \delta^{(hi)}, t_k + \phi]$. The upper bound for these two cases is quantified by Equation 7.16.

To see that Equation 7.17 is an upper bound on the carry-in from mode $M^{(h)}$ to $M^{(i)}$, observe that if there are no deadline misses prior to mcr_{k-1}, then all jobs with deadlines prior to t_{k-1} have completed execution. Thus, only (non-aborted) higher priority jobs of $\tau^{(h)}$ can contribute to the carry-in to $M^{(i)}$. (Note the constraint of Equation 5.2 prevents carry-in jobs from previous mode changes). For higher or equal priority tasks of $\tau^{(hi)}$, the contribution of these tasks to $\mathsf{ci}(\mathsf{mcr}_{k-1})$ is maximized if each job of $\tau_\ell^{(hi)} \in \tau^{(hi)}$ arrives just prior to $t_{k-1} + \delta^{(hi)}$. This accounts for the first summation in Equation 7.17. For all tasks $\tau_\ell^{(h)} \in \vartheta^{(hi)}$, there is at most one job of $\tau_\ell^{(h)}$ active at time t_{k-1}. These jobs may contribute to the execution of $\mathsf{ci}(\mathsf{mcr}_{k-1})$ only if the total execution of $\vartheta^{(hi)}$ and jobs of $\tau^{(hi)}$ that may interfere over $[t_{k-1}, t_{k-1} + \delta^{(hi)})$ (given that the last job of $\tau_\ell^{(hi)}$ arrives just prior to $t_{k-1} + \delta^{(hi)}$) is greater than the total supply over the transition period. This accounts for the term inside the large $()_+$ in Equation 7.17.

To see that Equation 7.18 is also an upper bound on the carry-in from mode $M^{(h)}$ to $M^{(i)}$, observe that each job carried-in (according to Definition 14) from $M^{(h)}$ to $M^{(i)}$ must have priority at least as τ_f; therefore, by definition Ψ in Equation 7.16, $\Psi(\mathsf{mcr}_{k-1}, \delta_{hi}, \tau_f)$ is clearly an upper bound on the execution requests of jobs of $\tau^{(h)} \setminus \alpha^{(hi)}$ over the interval $[t_{k-1}, t_{k-1} + \delta^{(hi)}]$. We subtract the minimum execution received over $[t_{k-1}, t_{k-1} + \delta^{(hi)}]$ to obtain the carry-in that is the first part inside the $\max$ function in Equation 7.18. However, this carry-in is lower bounded by the total higher priority unchanged tasks (i.e., $\sum_{\tau_\ell^{(h)} \in \mathsf{hp}(\tau^{(h)}, \tau_f)} e_\ell^{(h)}$). This is due to all higher-priority unchanged tasks may release jobs right before the transition period ends (i.e., $t_{k-1} + \delta_{hi}$). ■

To prove Lemma 23, we need two additional helper lemmas. The next lemma shows that $E_{ij}(\tau_f)$ (i.e., Equation 7.21) is an upper bound on the amount of carry-in (with priority higher or same as τ_f) from mode $M^{(i)}$ to $M^{(j)}$. The proof is similar to the second part of the Lemma 22.

Lemma 24 *For any sequence of mode changes $\mathsf{mcr}_0, \mathsf{mcr}_1, \mathsf{mcr}_2, \ldots$, consider a mode change from $M^{(i)}$ to $M^{(j)}$ (i.e., $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$). If the actual carry-in (with priority higher or same as τ_f) from $M^{(i)}$ to $M^{(j)}$ is greater than $E_{ij}(\tau_f)$, then some job generated in mode $M^{(i)}$ or during the transition (i.e., between t_k and $t_k + \delta_{ij}$) missed a deadline before $t_k + \delta_{ij}$.*

We prove this lemma by contradiction. The proof is similar to the bound in Equation 7.17. Assume that the carry-in $\mathsf{ci}(\mathsf{mcr}_k)$ is greater than $E_{ij}(\tau_f)$, but no job of $M^{(i)}$ misses a deadline. Only (non-aborted) jobs of $\tau^{(i)}$ with priority at least as τ_f can contribute to the carry-in to $M^{(i)}$, since $d_\ell^{(i)} \leq p_\ell^{(i)}$ for all $\tau_\ell^{(i)} \in \tau^{(i)}$. (Note the constraint of Equation 5.2 prevents carry-in jobs from previous mode changes). For all tasks of $\vartheta^{(ij)}$, there is at most one such job; the total execution of these jobs is $\sum_{\tau_\ell^{(i)} \in \mathsf{hp}(\vartheta^{(ij)}, \tau_f)} e_\ell^{(i)}$. For tasks $\tau_\ell^{(ij)}$ of $\tau^{(ij)}$, an upper bound on the execution that this task contributes to the carry-in is the execution requirement of $\tau_\ell^{(ij)}$ times the total number of jobs that can arrive in the interval $[t_k, t_k + \delta^{(ij)}]$ plus the execution of at most one job that can arrive prior to t_k. Adding together the total execution of all non-aborted jobs from $M^{(i)}$ and subtracting the minimum supply over the transition gives the upper bound of Equation 7.17. Thus, if more than $E_{ij}(\tau_f)$ execution is carried-in, then it must come from a job generated prior to t_k with remaining execution. However, by the above discussion, this is not possible unless a job misses a deadline.

In the next helper lemma, we establish two properties of the function Ψ described in Equation 7.24.

Lemma 25 *Given $M^{(i)}$, $M^{(j)}$, and $\phi, \zeta \in \mathbb{N}$, functions*

P1: $\Psi^{i,j}(\zeta, \phi, \tau_f)$, $E_{ij}(\tau_f)$, and $F_{ij}(\zeta, \tau_f)$ are integer-valued function.

P2: $\Psi^{i,j}(\zeta, \phi, \tau_f)$ and $F_{ij}(\zeta, \tau_f)$ are monotonically non-decreasing on ζ.

The property P1 always holds due to the fact that all task characteristics are natural integers. Furthermore, floor and ceiling functions are used for calculating ratios in the demand and supply functions. The second property follows from the fact that ζ is directly added in the definition of $\Psi^{i,j}(\zeta, \phi, \tau_f)$ depending on the value of $N^{(i)}$ and $\Pi^{(i)}$ which are independent of ζ; so, there is no way of getting lower value from the function $\Psi^{i,j}(\zeta, \phi, \tau_f)$ for any pair of modes $M^{(i)}$ and $M^{(j)}$ at a given ϕ with higher value of ζ than smaller value of ζ. Since $F_{ij}(\zeta, \tau_f)$ changes only with Ψ (the remainder is fixed with respect to ζ), F_{ij} is also monotonically non-decreasing.

For any pair of modes $M^{(i)}$ and $M^{(j)}$, the Equations 7.21 and 7.22 calculate the maximum carry-in for a mode-change from $M^{(i)}$ to $M^{(j)}$. Using these two equations, in the next lemma, we show that the carry-in after ℓ-th MCR is always bounded from above by $\mathsf{ci}_{\ell}^{i,j}(\tau_f)$.

Proof of Lemma 23 The proof is by induction on k. By definition, $\mathsf{ci}(\mathsf{mcr}_k, \tau_f)$ denotes the carry (with priority higher or same as τ_f) after the transition period past mcr_k.

Base Case: The base case is $k = 1$. To show that $\mathsf{ci}(\mathsf{mcr}_1, \tau_f) \leq \mathsf{ci}_1^{i,j}(\tau_f)$, we must show that both $\mathsf{ci}(\mathsf{mcr}_1, \tau_f) \leq E_{ij}(\tau_f)$ and $\mathsf{ci}(\mathsf{mcr}_1, \tau_f) \leq F_{ij}(0, \tau_f)$ are satisfied. Note that $\mathsf{ci}(\mathsf{mcr}_0, \tau_f)$ is equal to zero. By the fact that there are no deadline misses prior to t_1 and Lemma 24, the first condition is satisfied. For the second condition, $\Psi^{i,j}(\delta_{ij}, 0, \tau_f)$ by definition is greater or equal to the maximum execution request for an interval length of δ_{ij} after the MCR since the carry-in at the beginning for $M^{(i)}$ is zero. Therefore, the carry-in must be smaller than $\left(\Psi^{i,j}(0, \delta_{ij}, \tau_f) - \beta_{\mathsf{trans}}^{i,j}(\delta_{ij}), \sum_{\tau_\ell^{(i)} \in \mathsf{hp}(\tau^{(i)}, \tau_f)} e_\ell^{(i)}\right)$.

Induction hypothesis: Assume that the carry-in with equal or priority higher than τ_f is always less than $\mathsf{ci}_k^{i,j}(\tau_f)$ for any sequence of length k or less MCRs.

Induction: The $\mathsf{ci}^{i,j}_{k+1}(\tau_f)$ is the minimum of $E_{ij}(\tau_f)$ and $F_{ij} \overset{\text{def}}{=} \max_h F_{ij}(\mathsf{ci}^{h,i}_k(\tau_f), \tau_f)$. In case $\mathsf{ci}^{i,j}_{k+1}(\tau_f)$ is determined by $E_{ij}(\tau_f)$, the induction step follows trivially as $\mathsf{ci}(\mathsf{mcr}_{k+1}, \tau_f)$ must be always less or equal to $E_{ij}(\tau_f)$ for a schedulable subsystem (by Lemma 24). Thus, we must consider if $\mathsf{ci}^{i,j}_{k+1}(\tau_f)$ corresponds to F_{ij}. The function $F_{ij}(\zeta, \tau_f)$ is monotonically non-decreasing on ζ. By induction hypothesis, $\mathsf{ci}(\mathsf{mcr}_k, \tau_f) \le \mathsf{ci}^{i,j}_k(\tau_f)$. Thus, $\max\limits_{h=1,\ldots,q \wedge h \neq i} \mathsf{ci}^{h,i}_k(\tau_f) \ge \mathsf{ci}(\mathsf{mcr}_k, \tau_f)$. Beside, by definition $\max\limits_{h=1,\ldots,q \wedge h \neq i} \Psi^{i,j}(\mathsf{ci}^{h,i}_k(\tau_f), \phi, \tau_f)$ is the upper bound on request with priority higher or same as τ_f for all $\phi \ge 0$ after a mode change to $M^{(i)}$. As $\beta^{i,j}_{\mathsf{trans}}(\delta_{ij})$ is the lower bound on the supply, $\max\limits_{h=1,\ldots,q \wedge h \neq i} \Psi^{i,j}(\mathsf{ci}^{h,i}_k(\tau_f), \phi, \tau_f) - \beta^{i,j}_{\mathsf{trans}}(\delta_{ij})$ is the upper bound on the carry which is exactly F_{ij}; therefore, $\mathsf{ci}^{i,j}_k(\tau_f)$ is clearly an upper bound on $\mathsf{ci}(\mathsf{mcr}_k, \tau_f)$. ■

Lemma 26 *For $\phi \ge 0$ and any $\mathsf{mcr}_k = (M^{(i)}, M^{(j)}, t_k)$ in arbitrary sequence of MCRs, if $\Psi^{i,j}(\phi, \zeta, \tau_f)$ is at least $\xi \ge 0$, then the value of x that maximizes the supremum in the right-hand-side of Equation 7.24 occurs at or before the maximum of $d^{(i)}_{\max}$ and the minimum of $\mathsf{lcm}_{\tau^{(i)}_\ell \in \tau^{(i)}}\{p^{(i)}_\ell\} + d^{(i)}_{\max}$ and*

$$\left\lceil \frac{u^{(i)}(\tau_f) \cdot p^{(i)}_{\max} + u^{(ij)}(\tau_f) \cdot \phi + \zeta + e^{(i)}_{\max} + \frac{\Theta^{(i)}(\Pi^{(i)} - \Theta^{(i)})}{\Pi^{(i)}} + \sum_{\tau^{(i)}_\ell \in \mathsf{hp}(\tau^{(i)}, \tau_f)} e^{(i)}_\ell - \xi}{\frac{\Theta^{(i)}}{\Pi^{(i)}} - u^{(i)}(\tau_f)} \right\rceil \tag{7.29}$$

where $u^{(i)}(\tau_f) \overset{def}{=} \sum\limits_{\tau^{(i)}_\ell \in \mathsf{hp}(\tau^{(i)}, \tau_f)} u^{(i)}_\ell$, and $p^{(i)}_{\max} \overset{def}{=} \max\limits_{\tau^{(i)}_\ell \in \tau^{(i)}} \{p^{(i)}_\ell\}$.

Proof: When $u^{(i)}(\tau_f)$ is at most $\Theta^{(i)}/\Pi^{(i)}$, it may be shown via techniques similar to Baruah et al. [9] that $\mathsf{lcm}_{\tau^{(i)}_\ell \in \tau^{(i)}}\{p^{(i)}_\ell\} + d^{(i)}_{\max}$ is an upper bound on x. However, when $u^{(i)}(\tau_f)$ is strictly less than $\Theta^{(i)}/\Pi^{(i)}$ we may obtain a potentially tighter upper bound. Suppose that the right-hand side of Equation 7.24 obtains its supremum at some $x > d^{(i)}_{\max}$. According to Equation 7.24, we have

$$\begin{aligned}
\xi \quad < & \sum_{\tau_\ell^{(i)} \in \mathsf{hp}(\vartheta^{(ij)}, \tau_f)} \left(\left\lfloor \frac{x}{p_\ell^{(i)}} \right\rfloor + 1 \right) \times e_\ell^{(i)} \\
& + \sum_{\tau_\ell^{(i)} \in \mathsf{hp}(\tau^{(ij)}, \tau_f)} \left(\left\lfloor \frac{x+\phi}{p_\ell^{(i)}} \right\rfloor + 1 \right) \times e_\ell^{(i)} \\
& + \sum_{\tau_\ell^{(i)} \in \mathsf{hp}(\alpha^{(ij)}, \tau_f)} \left\lfloor \frac{x}{p_\ell^{(i)}} \right\rfloor \times e_\ell^{(i)} \\
& + \sum_{\tau_\ell^{(i)} \in \mathsf{hp}(\alpha^{(ij)}, \tau_f)} \min \left(x - \left\lfloor \frac{x}{p_\ell^{(i)}} \right\rfloor p_\ell^{(i)}, e_\ell^{(i)} \right) \\
& + \zeta + e_{\max}^{(i)} - \beta_{\mathsf{prior}}^{i}(x) \\
\\
< & \sum_{\tau_\ell^{(i)} \in \mathsf{hp}(\vartheta^{(ij)}, \tau_f)} \frac{e_\ell^{(i)}}{p_\ell^{(i)}} \times x \\
& + \sum_{\tau_\ell^{(i)} \in \mathsf{hp}(\tau^{(ij)}, \tau_f)} \frac{e_\ell^{(i)}}{p_\ell^{(i)}} \times (x+\phi) \\
& + \sum_{\tau_\ell^{(i)} \in \mathsf{hp}(\alpha^{(ij)}, \tau_f)} \frac{x}{p_\ell^{(i)}} \times e_\ell^{(i)} \\
& + \sum_{\tau_\ell^{(i)} \in \mathsf{hp}(\alpha^{(ij)}, \tau_f)} e_\ell^{(i)} + \sum_{\tau_\ell^{(i)} \in \mathsf{hp}(\tau^{(i)}, \tau_f)} e_\ell^{(i)} \\
& + \zeta + e_{\max}^{(i)} - \beta_{\mathsf{prior}}^{i}(x)
\end{aligned}$$

$$\begin{aligned}
\Rightarrow 0 \quad & < x \cdot u^{(i)}(\tau_f) + \phi \cdot u^{(ij)}(\tau_f) + u^{(i)}(\tau_f) \cdot \max \left\{ p_\ell^{(i)} \right\} + e_{\max}^{(i)} \\
& + \sum_{\tau_\ell^{(i)} \in \mathsf{hp}(\tau^{(i)}, \tau_f)} e_\ell^{(i)} - x \frac{\Theta^{(i)}}{\Pi^{(i)}} + \frac{\Theta^{(i)}}{\Pi^{(i)}} (\Pi^{(i)} - \Theta^{(i)}) - \xi + \zeta
\end{aligned}$$

Solving for x and noting that x must be an integer implies the upper bound of Equation 7.29. ■

CHAPTER 8: RESOURCE ESTIMATION OF MMS

In previous chapters, we assumed that subsystems are sharing a common hardware platform, and coupled each software mode to a hardware mode for ensuring schedulability and the temporal isolation among subsystems. We exploited the coupling of software and hardware modes to develop a pseudo-polynomial algorithm for the problem of schedulability analysis of multi-modal real-time systems. This pseudo-polynomial schedulability-test effectively decoupled the schedulability analysis from choosing appropriate hardware resource for each software mode. As the development of a new system usually starts with requirement specification, the software modes may be chosen directly from the specification and from the nature of workload execution. The schedulability of these software modes can be checked by assigning the highest hardware resource to each mode, then invoke the schedulability analysis developed in the previous section. Checking schedulability for real-time systems may not be ultimate goal pursued by the real-time system designer; an optimized schedulable real-time multi-modal system is always preferred over barely schedulable system. In this chapter, we address the resource usages of multi-modal system and achieve the notion of optimality for a MMS.

8.1 Motivation

In this section, we address how a multi-modal system can be optimized with respect to hardware resource usages. First, we discuss about the source of unoptimized result. Notice that if the system has ρ number of hardware modes against the total q number of subsystem modes the designer tries to achieve, the total number of combinations could be q^{ρ}. We address each combination as a configuration. The number of hardware modes depends on the underlying modeling techniques. Consider an EDP-resource Ω with resource-period Π. Using resource-period Π, there could be

Π hardware modes. Each of q modes of a configuration can take each of Π hardware modes; therefore, the total number of configurations can be as much as $O(q^{\Pi})$. The number of configurations could be much higher in case we choose a different resource deadline (Δ) than the resource capacity. Among this exponential number of configurations, only a handful of configurations may be valid. For instance, a system might not support the most computationally-expensive execution pattern for the lowest energy mode. We can check the schedulability for each of these configurations by invoking SUBI with desirable $N^{(i)}$ parameter. This exponential number of configurations is not limited to only EDP-resources, similar situation is not uncommon for other available modeling techniques (i.e. TDMA server, DVFS). The problem may arise due to the granularity in choosing hardware modes.

8.2 Notion of Optimality

We first determine the notion of optimality with respect to the objective function (e.g. minimizing weighted sum of capacities of modes, minimizing the maximum capacity over all modes). To the best of our knowledge, none of the previous results on real-time multi-modal systems address the optimality in terms of resource usages with respect to any objective function. We exploit the schedulability analysis developed in previous sections to obtain set of optimized hardware resources ensuring system schedulability. A very naive solution could try all possible hardware/software configurations (depicted in Figure 8.1) and use the schedulability test derived in

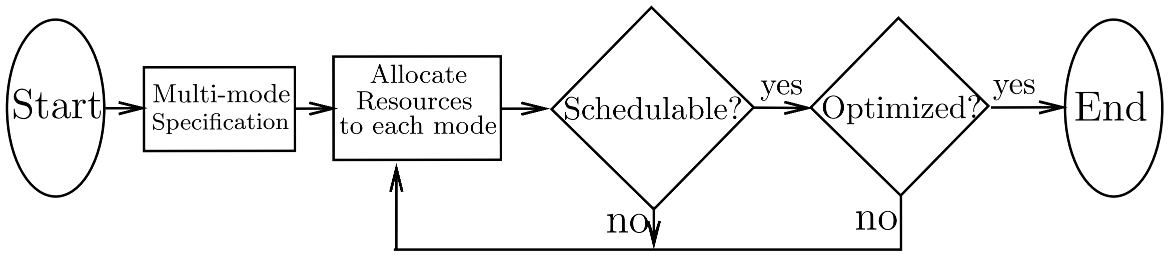

Figure 8.1: Resource optimization for multi-modal systems.

the previous chapters to find the optimal with respect to the corresponding objective function. The process of determining optimization is also known design-space exploration [3, 78] in the context of system design; a naive solution is depicted in Figure 8.1. We believe that the complexity for determining optimized set of modes depend on the objective function. Our future work will address the more general objective, minimizing weighted sum of capacities over all modes, for which all configurations may need to check for optimality.

8.3 Minimizing the Maximum Resource

In this section, we address the objective function of minimizing the maximum resource usages for a set of software modes, and develop an optimal solution with with pseudo-polynomial time complexity for the objective function. A naive approach may be the repeated application of SUBI with varying capacity for each application mode (e.g., depicted in Figure 8.1). As different values of $\Theta^{(i)}$ generate different subsystems, we denote each subsystem by $\mathcal{S}_k \overset{\text{def}}{=} \langle\tau$, τ^{trans}, ϑ, α, δ, $N, \Omega_k\rangle$ where $\Omega_k \overset{\text{def}}{=} \{\Omega_k^{(1)} \ldots \Omega_k^{(q)}, \ldots, \Omega_k^{(ij)}, \ldots\}$, $\Omega_k^{(i)} \overset{\text{def}}{=} (\Pi^{(i)}, \Theta_k^{(i)}, \Delta^{(i)})$, and $\Delta^{(i)} \overset{\text{def}}{=} \Theta_k^{(i)}$. In this notation, τ represents the vector of task systems (i.e., $\tau \overset{\text{def}}{=} [\tau^{(1)}, \ldots, \tau^{(q)}]$); τ^{trans} is a matrix of task sets representing the unchanged tasks between any two modes; ϑ and α are matrices of task sets representing respectively the finished tasks and the aborted tasks when transitioning from mode $M^{(i)}$ to $M^{(j)}$; δ is a matrix of transition offsets; and N is a vector indicating the minimum number of resource periods between two mode changes (i.e., $N \overset{\text{def}}{=} [N^{(1)}, \ldots, N^{(q)}]$). We denote $\mathcal{S}(\tau, \tau^{\mathsf{trans}}, \vartheta, \alpha, \delta, N, \Omega)$ as the set of all such subsystems $\mathcal{S}_k$; note that all the parameters except the resource capacities are identical in $\mathcal{S}$. The size of $\mathcal{S}(\tau, \tau^{\mathsf{trans}}, \vartheta, \alpha, \delta, N, \Omega)$ can be $O((\max_{i\in\{1,\ldots,q\}} \Pi^{(i)})^q \times (\max_{i,j\in\{1,\ldots,q\}} \Pi^{(ij)})^{q^2})$ which is exponential. In this exponential search space, the total number of candidates may be restricted based on objective functions (e.g. minimizing the maximum resource over all subsystem modes).

Multi-modal systems developed so far have not addressed the optimal (hardware) resource allocation for each mode. Optimal solutions may vary due to different objective functions (e.g., minimizing peak-temperature or minimizing the total energy consumption). We address the objective function of minimizing the maximum resource usages, denoted as *MinMax Resource*, for a set of application modes. In Chapter 4, we have shown that minimizing the capacity of a periodic resource is useful for minimizing the peak-system temperature [6] in a system with simple active/idle power modes. Our future work will address a more general objective of *minimizing weighted sum of capacities over all modes* for which all combinations of resource usages may need to be evaluated. Rest of the paper use the notation $\mathcal{S}_k^{(i)}$ to denote a mode in $\mathcal{S}_k$, and evaluate the inequality $\Omega_k^{(i)} \succeq \Omega_k^{(j)}$ by the expression $I_k^{(i)} \geq I_k^{(j)}$. The following observation describes a favorable characteristic of the SUBI algorithm:

Observation 1 *Given τ, τ^{trans}, δ, ϑ, α, Ω, and N, for any two subsystems $\mathcal{S}_1, \mathcal{S}_2 \in \mathcal{S}(\tau, \tau^{\mathsf{trans}}, \vartheta, \alpha, \delta, N, \Omega)$ and $\forall i$: $\Omega_1^{(i)} \preceq \Omega_2^{(i)}$, if* $\mathsf{SUBI}(\mathcal{S}_2)$ *returns* NO*, then* $\mathsf{SUBI}(\mathcal{S}_1)$ *will also return* NO*.*

The observation says that if SUBI returns NO response to a resource combination, we must increase resource (i.e., bandwidth) for at least one mode for schedulability. This observation can be further extended to achieve monotonicity for the SUBI algorithm over a subset of $\mathcal{S}$. Given τ, τ^{trans}, ϑ, δ, α, and N, consider an ordered set $\{\mathcal{S}_k | k \in \mathbb{N}_+\}$ where $\forall_{k,i} \Omega_k^{(i)} \preceq \Omega_{k+1}^{(i)}$. In other words, this is a totally ordered subset of $\mathcal{S}$. According to Observation 1, if the SUBI returns NO for any $\mathcal{S}_k$, then the algorithm also returns NO for all preceding elements before $\mathcal{S}_k$ which is formalized as follows:

Lemma 27 *For any given τ, ϑ, α, δ, and Π;* SUBI *is monotonically non-decreasing over a totally-ordered set of subsystems $\{\mathcal{S}_k | \mathcal{S}_k \in \mathcal{S}(\tau, \tau^{\mathsf{trans}}, \vartheta, \alpha, \delta, N, \Omega) \wedge k \in \mathbb{N}_+\}$ where $\forall_{k,i} \Omega_k^{(i)} \preceq \Omega_{k+1}^{(i)}$.*

The lemma follows from the fact that each of the service functions (i.e., β and sbf) is monotonically increasing on $\Omega^{(i)}$ and the demand functions (i.e., cidbf) are monotonically decreasing on $\Omega^{(i)}$. Thus, if conditions of Theorem 9 hold for a small value of $\Theta^{(i)}$, conditions will continue to hold for larger values of $\Theta^{(i)}$. Using this monotonicity, we develop an algorithm for determining an optimized multi-mode system.

Algorithm 3 MinMaxCap($\tau, \tau^{\mathsf{trans}}, \vartheta, \alpha, \delta, N, \Omega$).

```
1: {Initialization}
2: Set all $\Omega^{(i)}$ and $\Omega^{(ij)}$ to max bandwidth 1 for all $1 \leq i, j \leq q$ (e.g., set $\Theta^{(i)}$ to $\Pi^{(i)}$)
3: Check schedulability and return $\emptyset$ for an unschedulable result.
4: Set $I_{\text{success}}$ to 1
5: unoptimized $\leftarrow \{1 \ldots q\}$
6: {Optimization loop starts from here.}
7: repeat
8:     Use a binary search to determine the minimum $I_{\text{success}} \in [\max u^{(i)}, I_{\text{success}}]$ which makes all
       modes in unoptimized schedulable.
9:     Determine optimized by checking for each element of unoptimized if reducing the capacity by
       one results in unschedulability
10:    Update $\Omega^{(i)}$, $\Omega^{(ij)}$, and $\Omega^{(ji)}$ according to $I_{\text{success}}$ for $i \in$ optimized
11:    Remove all modes in optimized from unoptimized
12: until unoptimized is empty OR remains unchanged from previous iteration
13: return $\mathcal{S}$
```

8.3.1 The MinMaxCap algorithm

The pseudocode for the MinMaxCap is presented in Algorithm 3. The total number of application modes is q. The algorithm starts with the highest bandwidth for each application mode, and then optimizes all the subsystem modes gradually. Unoptimized modes are tracked by the set **unoptimized** which is initialized by all mode indexes. Modes in the complement of the set **unoptimized** have already optimized resource usages. The algorithm continues until the set **unoptimized** is empty or remained unchanged from the previous step. Using a binary search technique, the algorithm determines the minimum I_{success} for which all modes in **unoptimized** are schedulable. The algorithm then updates **unoptimized** by determining the set **optimized** for which further reduction in the bandwidth would result unschedulability. This could be accomplished by iterating each mode index i in **unoptimized**, and setting capacity $\Omega^{(i)}$ to I_{success}, but all other resources to $\lfloor (I_{\text{success}} \times \Pi^{(i)} - 1) \rfloor / \Pi^{(i)}$, then, invoke SUBI, and include i in **optimized** if the result is true. The total cost of this step is no more than $O(q)$ invocations of SUBI.

8.3.2 Complexity and Correctness

The algorithm $\mathsf{MinMaxCap}$ optimizes resource usages of a multi-mode system where system parameters are denoted by τ, τ^{trans}, ϑ, δ, α, Ω, and N. To formalize, we prove the following theorem:

Theorem 8 *Given τ, τ^{trans}, ϑ, α, Ω, δ, and N, the procedure* $\mathsf{MinMaxCap}(\tau, \tau^{\mathsf{trans}}, \vartheta, \alpha, \delta, N, \Omega)$ *returns $\mathcal{S}_*$ such that $\mathcal{S}_*$ is* EDF*-schedulable and $\max\limits_i \Omega_*^{(i)}$ is the minimum among all schedulable multi-mode systems with respect to* SUBI*, i.e.,*

$$\max_i I_*^{(i)} \leq \max_i I_k^{(i)}$$

where $\mathcal{S}_k$ denotes any schedulable subsystem. Furthermore, $\mathsf{MinMaxCap}(\tau, \tau^{\mathsf{trans}}, \vartheta, \alpha, \delta, N, \Omega)$ *has time complexity $O\left(Tq^2 \lg \Pi\right)$ where T denotes the time complexity of the* SUBI *algorithm and Π is the maximum resource period.*

Proof: The proof is by contradiction. Assume that the returned multi-modal system $\mathcal{S}_*$ from $\mathsf{MinMaxCap}$ is not optimal with respect to the minimum maximum bandwidth over all modes; there is another schedulable $\mathcal{S}_o$ for which the subsystem mode with the highest bandwidth is minimized. That is $\max_i I_*^{(i)} > \max_i I_o^{(i)}$. Let us assume $I \overset{\text{def}}{=} \max_i I_o^{(i)}$. Observe that unoptimized modes comprise the set **unoptimized**. Initially the set **unoptimized** equals $\{1, \ldots, q\}$. For **unoptimized**, the algorithm finds the minimum I_{success} using a binary search for which setting bandwidth to I_{success} make the system schedulable. The inequality $I > \max_i I_*^{(i)}$ implies that the returned I_{success} is greater than I; that is, the algorithm did not find any $I' < I_{\text{success}}$ for which setting resource of each mode in **unoptimized** to I' results schedulability. As $I < I_{\text{success}}$, the system $\mathcal{S}_o$ cannot be schedulable (Observation 1) which is a contradiction.

To see the complexity is $O(Tq^2 \lg \Pi)$, observe that Line 9 finalizes at least one mode at each iteration. At each iteration, binary search requires the total $\lg \Pi$ invocations of the algorithm SUBI due to integer system parameters assumption. To determine the set optimized at Line 9,

Table 8.1: Tasks Distribution in Modes.

Tasks	Properties			$Modes$		
#	$e^{(i)}$	$d^{(i)}$	$p^{(i)}$	$M^{(1)}$	$M^{(2)}$	$M^{(3)}$
1	1	10	10	y	y	y
2	5	30	30	y	y	n
3	4	40	40	n	n	y
4	1	10	10	y	n	n
5	1	20	20	n	y	y
6	3	24	24	n	n	y
7	2	20	20	y	n	n
8	1	10	10	n	y	n

the algorithm requires at most q invocations of SUBI. As there are q number of modes, in the worst case, the algorithm invokes SUBI at most $O(q^2 \lg \Pi)$ times which result the complexity to be $O(Tq^2 \lg \Pi)$. ■

We optimized resource capacity; similarly one could apply this technique for any other parameters (e.g., transition period).

8.4 Simulations

In this section, we present the performance results for our proposed algorithm. We compare SUBI with exponential-time *schedulability analysis using reachability graph* (SURG) proposed by Phan et al. [63]. For the simulation, we implemented SURG and SUBI in MATLAB and performed our simulations on a 2.33GHz Intel Core 2 Duo machine with 2.0GB RAM.

We performed set of experiments using tasks described in Table 8.1. We compare MinMaxCap with the algorithm SURG by [63]. While checking schedulability, the SURG eventually calculates the required resource for each mode. Each multi-mode subsystem consists of 8 modes where each mode select tasks randomly. The $\Pi^{(i)}$ is set to 10 for all modes. We measure the maximum capacity returned from both algorithms. In Figure 8.2, we plot the maximum mode utilization for subsystems in the horizontal axis. For each of the maximum mode utilizations, we generated at least 15 schedulable multi-mode systems, and plotted the average maximum capacity in the vertical axis. Figure 8.2 substantiates that MinMaxCap minimizes the resource usages over the

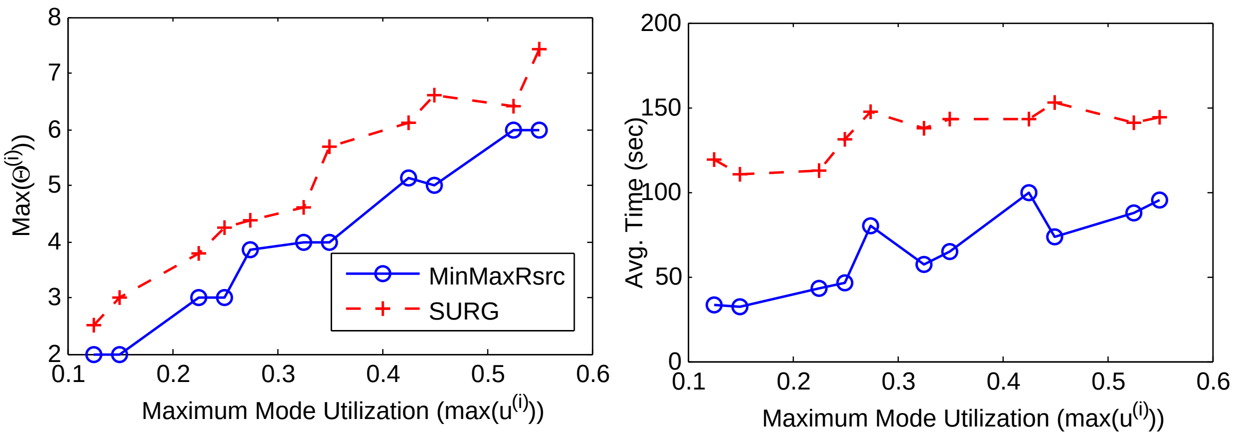

Figure 8.2: Comparison of SURG and SUBI: Resource usages

SURG and is clearly time efficient. Here, it is to be noted that SURG is more general, and can support additional features (e.g., input/output buffer, arbitrary mode-change request); so the comparison is only valid for restricted settings of control systems. Using a fundamentally different approach, we exploited these restricted settings to develop a pseudo-polynomial algorithm over the previous algorithm.

8.5 Conclusion

In this chapter, we introduced an efficient algorithm for allocating resources to multi-mode real-time systems. Simulation results validate the effectiveness and efficiency of our algorithms. Our future work will address the general objective of minimizing energy consumption for real-time systems supporting multiple application/resource modes.

CHAPTER 9: PARALLEL SCHEDULABILITY ANALYSIS

In order to ensure temporal isolation and also hard deadlines, the schedulability analysis developed in the previous chapter requires coupling each software mode to a hardware mode. There could be a very large number of software and hardware mode combinations for a given set of software and hardware modes. Our approach in the previous chapter is an effort to reduce the overall time complexity to less than exponential time. Even with this reduction, the algorithm requires a significant amount of time to decide the schedulability when the number of modes in the system is large. Besides, for real-time control systems, a larger number of modes is typically desirable to permit greater adaptability in unfavorable environments. Checking schedulability of multi-modal systems warrants higher computation time as a result of dependencies (due to transitions) between modes; therefore, a large number of modes pose a computational challenge for existing sequential schedulability analysis techniques. Thus, parallel schedulability analysis is a promising and practical alternative to traditional sequential schedulability analysis for multi-modal systems.

An efficient parallel schedulability analysis can also reduce significantly the time for resource estimation (also known as *design-space exploration* [78]) that may utilize schedulability tests for determining optimized resource parameters of a multi-mode system as mentioned in Section 8.2. Schedulability analysis using parallel algorithms is a relatively unexplored area for multi-modal real-time applications. For uni-modal systems, there have been solutions with well defined sets of conditions where each condition must pass a set of test cases. In most scenarios, the evaluation of these test case elements can be performed independently. From the perspective of parallel computing, this independent execution behavior makes the problem of uni-modal schedulability less challenging. However, the schedulability analysis of multi-modal real-time systems is complex; the analysis not only depends on each mode itself, but also on the schedulability of all other modes along with mode change sequences. Therefore, pseudo-polynomial schedulability analysis developed in the previous chapter may be a workable solution for the schedulability of systems (given a set of system parameters). However, a sequential implementation of the analysis may not sufficiently scale to be used as an effective tool for determining optimal system parameters (i.e., Chapter 8). In this thesis, we address a fundamental gap in the research literature on parallel schedulability analysis algorithms suitable for design-space exploration (e.g., minimizing the total aggregate hardware resources over all modes) for real-time multi-modal systems.

In this chapter, we propose the parallel algorithm for checking schedulability of multi-modal systems suitable for suitable for a computing cluster (i.e., High Performance Computing Grid at Wayne State University). This algorithm can be implemented using message passing interface (MPI), open multi-processing (OpenMP API), or win/linux thread library (e.g., pthread). Finally, we extend the parallel algorithm suitable for massive parallel computing and cost effective GPU platforms. This chapter develops parallel algorithm only for checking EDF-schedulability. For FP schedulability analysis of a MMS, one may develop identical parallel algorithm using techniques outlined in this chapter. We first describe performance metrics in the following section used for measuring performance of our proposed parallel schedulability analysis.

9.1 Parallel Performance Metrics

The asymptotic performance of a parallel algorithm is measured by using well known parallel performance metrics which include *parallel execution time*, *speedup*, *efficiency*, and *cost* [39].

§**Parallel Execution Time.** The parallel execution time, denoted by T_m, is the time elapsed between the start and the end of a parallel computation. The value of T_m depends on the actual workload, number of processors m, and the parallel overhead. T_m decreases at a slower rate as m increases. However, the parallel overhead also increases with m; therefore, after certain value of m, T_m may not experience a noticeable decrease.

§**Speedup.** The speedup S is defined as the ratio of the serial execution time T_s of the best sequential algorithm to the parallel execution time. That is, $S \overset{\text{def}}{=} \frac{T_s}{T_m}$. The perfect speedup for a parallel algorithm equals to m which may be difficult to achieve for algorithms that require communication/synchronization for the correct operation. Due to overhead, S may decrease as m increases.

§**Efficiency.** The Efficiency, denoted by E, is a measure of the fraction of time for which the processors are usefully employed in solving the problem; it is defined as the ratio of the speedup to the number of processors, $E \overset{\text{def}}{=} \frac{S}{m}$. The efficiency accounts for the parallel overhead, and usually decreases as m increases.

§**Parallel Execution Cost.** The cost of a parallel algorithm is the product of the parallel execution time and the number of processors.

9.2 Parallel Schedulability Analysis

Our proposed solution is the *first non-trivial* parallel schedulability analysis algorithm for real-time multi-mode systems. We provide a parallel algorithm for evaluating the schedulability conditions developed in the previous chapter. Our contributions in this chapter are as follows:

- To achieve a balanced workload distribution, we adopt the most suitable and relatively simple workload distribution policy considering the nature of the conditions (Section 9.3.2). A balanced distribution of workload (without introducing overhead) is the key for achieving better speedup.

- To achieve a near-ideal speedup (equal to the number of processing nodes), we design the algorithm such that the communication and synchronization overhead is minimized.

- To characterize the effectiveness of the proposed algorithm, we determine parallel performance metrics (e.g., speedup, efficiency, and cost) and derive conditions to obtain better speedup (Section 9.3.3).

- To substantiate the performance of the proposed algorithm, we perform experiments upon a cluster of AMD Opteron computers (Section 9.3.4). We obtain high parallel efficiency (over 90%) which establishes that the proposed algorithm can be used as an efficient schedulability test for design-space exploration of real-time multi-mode systems.

9.2.1 Schedulability Conditions

In this section, we develop a parallel algorithm for checking schedulability conditions developed in Chapter 6. Those conditions are suitable for checking schedulability sequentially. For example, Ψ function needs to evaluate all possible busy interval lengths in previously executing mode. To calculate Ψ parallely, we separate x from Ψ so that each processing node of a cluster can as follows:

$$\Psi(M^{(i)}, M^{(j)}, \zeta, \phi) \overset{\text{def}}{=} \sup_{x>0}\{\Psi_x(M^{(i)}, M^{(j)}, \zeta, \phi)\}, \tag{9.1}$$

Using Ψ_x functions, we rephrased these schedulability conditions which enable us to evaluate each of these conditions parallely with a workload distribution. Over any possible (legal) sequence of mode-change requests, the system is EDF-schedulable, if the following five conditions hold for any two distinct modes $M^{(i)}$ and $M^{(j)}$,

$$\begin{aligned}
&\mathsf{SC}_1: \quad \sum_{\tau_\ell^{(j)} \in \tau^{(j)}} \mathsf{dbf}(\tau_\ell^{(j)}, t) \le \mathsf{sbf}(\Omega^{(j)}, t), \qquad \forall t \in \Upsilon^{\mathsf{SC}_1(j)};\\
&\mathsf{SC}_2: \quad \sum_{\tau_\ell^{(j)} \in \tau^{(j)} \setminus \tau^{(ij)}} \mathsf{dbf}(\tau_\ell^{(j)}, t) + \sum_{\tau_\ell^{(ij)} \in \tau^{(ij)}} \mathsf{dbf}(\tau_\ell^{(ij)}, t+s)\\
&\qquad\qquad \le \beta^{i,j}_{\mathsf{post}}(s,t),\\
&\qquad\qquad\qquad \forall s,t : (0 < s \le \delta_{ij}) \wedge (t \in \Upsilon^{\mathsf{SC}_2(i,j,s)});\\
&\mathsf{SC}_3: \quad \sum_{\tau_\ell^{(ij)} \in \tau^{(ij)}} \mathsf{dbf}(\tau_\ell^{(ij)}, t) \le \mathsf{sbf}(\Omega^{(ij)}, t), \forall t \in \Upsilon^{\mathsf{SC}_3(i,j)};\\
&\mathsf{SC}_4: \quad \Psi_x(M^{(i)}, M^{(j)}, C_i, \phi) \le \beta^{i,j}_{\mathsf{trans}}(\phi),\\
&\qquad\qquad\qquad \forall \phi, x : (0 < \phi \le \delta_{ij}) \wedge (x \in \Upsilon^{\mathsf{SC}_4(i,j,\phi)});
\end{aligned} \tag{9.2}$$

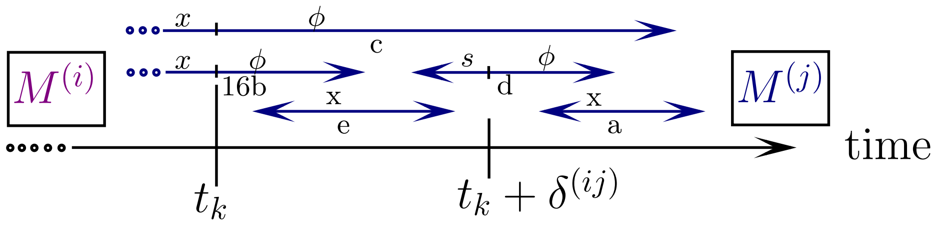

Figure 9.1: Busy intervals for each of the schedulability conditions SC_Z.

$$\begin{aligned}\mathsf{SC}_5 : \sum_{\tau_\ell^{(j)} \in \tau^{(j)} \setminus \tau^{(ij)}} &\mathsf{dbf}(\tau_\ell^{(j)}, t) + \Psi_x(M^{(i)}, M^{(j)}, C_i, \delta_{ij} + t) \\ &-\beta_{\mathsf{trans}}^{(,M}{}^{(i)}, M^{(j)}, \delta_{ij}) \leq \beta_{\mathsf{post}}^{i,j}(0, t), \\ &\forall t, x : (0 < t \leq \mathcal{T}_{ij}) \wedge (x \in \Upsilon^{\mathsf{SC}_5(i,j,t)});\end{aligned}$$

where $C_i \overset{\text{def}}{=} \max_{\{h=1,\ldots,q\} \wedge h \neq i} \{\mathsf{ci}(M^{(h)}, M^{(i)})\}$. $\Upsilon^{\mathsf{SC}_1(i)}$, $\Upsilon^{\mathsf{SC}_2(i,j,s)}$, $\Upsilon^{\mathsf{SC}_4(i,j,\phi)}$, $\Upsilon^{\mathsf{SC}_5(i,j,t)}$, and $\mathcal{T}_{ij}$ are each a finite set of consecutive positive integers starting from one. Each of these sets are commonly referred as a *testing-set*. We use a generic notation of $\mathsf{SC}_Z(i, j, \phi)$ where $Z \in \{1, \ldots, 5\}$ for the superscript of the testing sets. For example, the $\mathsf{SC}_1(i, \emptyset, \emptyset)$ is the superscript for $\Upsilon^{\mathsf{SC}_1(i)}$ which is the testing set of SC_1 in Equation 9.2. The last two parameters in this example have the value of $\emptyset$ as they are not used by SC_1. The number of testing sets associated with modes $M^{(i)}$, $M^{(j)}$, and schedulability condition SC_Z is denoted by $\text{TS}_Z(i, j)$. The testing set bounds have been proven in Chapter 6 (i.e., Fisher and Ahmed [35]). $\mathcal{T}_{ij}$ is specified by the number of $\text{TS}_5(i, j)$ sets that exist and we will not refer to $\mathcal{T}_{ij}$ further.

Now, we provide intuitive explanations for each condition of Equation 9.2. Before missing a deadline by an EDF-schedule, the processor is continuously busy. This interval is known as a busy interval. In the busy interval, the resource demand is greater than the processing supply. Five conditions are used to avoid busy intervals where resource demand is greater than the supply taking mode changes into account. We identified the five different kinds of busy intervals with respect to a mode change request, which are depicted in Figure 9.1. SC_1 ensures the schedulability of an individual mode. SC_2 and SC_3 ensure that an individual mode is schedulable along with the demand from unchanged tasks of the old mode after a mode change. SC_4 ensures the schedulability during the transition period while accounting for the carry-in demand from the non-aborted jobs and the mode-change supply function. Similarly, SC_5 ensures schedulability after a transition. The last two conditions account for the carry-in from all past mode change requests through the ci function while analyzing demand of each individual mode. Following section develop a parallel algorithm suitable for MPI.

9.3 Parallel Schedulability Using Message Passing

9.3.1 Processing Platforms

We consider a *parallel message-passing system* composed of m identical processors, $P = \{P_1, \ldots, P_m\}$ where the subscript $i \in \{1, \ldots, m\}$ for each processor P_i denotes the unique identifier (frequently denoted as *rank*) in the platform. The design of our parallel algorithm considers the data parallel model in which the total workload (testing-set elements) is statically mapped onto processors and each processor performs similar operations on different testing-set elements. For communication/synchronization, we use the parallel message-passing construct All-to-All-Reduction [39] by which all processors simultaneously involve in a communication/synchronization operation. The All-to-All-Reduction uses an associative operator (e.g., MAX, SUM, OR) to accumulate and combine the data from the buffer of each processor into a single piece of data which is then replicated at all processors.

9.3.2 Problem Formulation

We design a parallel algorithm for solving the EDF-Multi-Mode-Sched Problem. We first determine the complexity of a serial algorithm for checking all five conditions to realize the size of the problem to be parallelized. The runtime complexity depends on the total aggregate size of all the testing sets. Furthermore, conditions SC_4 and SC_5 require evaluating the ci function to account for the carry-in execution. In the previous chapter, we showed that the $\mathsf{ci}(M^{(i)}, M^{(j)})$ function, defined iteratively from the sequence $\mathsf{ci}_0^{M^{(i)},M^{(j)}}$, $\mathsf{ci}_1^{M^{(i)},M^{(j)}}$, $\mathsf{ci}_2^{M^{(i)},M^{(j)}}$,

... for all $i, j (i \neq j) \in \{1, \ldots, q\}$, can be calculated in a finite number of iterations which is equal to the summation of execution requirements of all tasks. We define $\mathcal{C}$ as the maximum of the following three values: 1) the summation of the execution requirements of all tasks over all modes, 2) the maximum transition period (i.e., maximum δ_{ij}) and 3) the maximum of the sizes of the sets $\Upsilon^{\mathsf{SC}_Z(i,j,\phi)}$ and $\mathcal{T}_{ij}$ for any i, j, ϕ and Z. The calculation of the $\mathsf{ci}(M^{(i)}, M^{(j)})$ for each pair requires at most $\mathcal{C}$ inductive computations of ci where each such computation would invoke the Ψ_x function at most $\mathcal{C}$ times. As there are $q(q-1)$ pairs of modes, a serial function for calculating the carry-in for all pairs would require $O(q^2 n\mathcal{C}^2)$ time, where n is $\max_{i=1}^{q}\{n_i\}$; the term n is due to the calculation of demand (dbf) for every testing set element. The complexity of checking all five conditions is dominated by the complexity of checking condition SC_5 which is also $O\left(q^2 n\mathcal{C}^2\right)$; therefore, the complexity of the serial schedulability analysis is $O\left(q^2 n\mathcal{C}^2\right)$. This pseudo-polynomial complexity could be quite large as $\mathcal{C}$ is potentially exponential in the representation of the multi-modal system. Thus, it is desirable to decrease the analysis time by parallelizing the schedulability analysis.

Parallel Platform

We consider a *parallel message-passing system* composed of m identical processors, $P = \{P_1, \ldots, P_m\}$ where the subscript $i \in \{1, \ldots, m\}$ for each processor P_i denotes the unique identifier (frequently denoted as *rank*) in the platform. The design of our parallel algorithm considers the data parallel model in which the total workload (testing-set elements) is statically mapped onto processors and each processor performs similar operations on different testing-set elements. For communication/synchronization, we use the parallel message-passing construct All-to-All-Reduction [39] by which all processors simultaneously involve in a communication/synchronization operation. The All-to-All-Reduction uses an associative operator (e.g., MAX, SUM, OR) to accumulate and combine the data from the buffer of each processor into a single piece of data which is then replicated at all processors.

The performance of a parallel algorithm depends heavily on the underlying workload distribution policy. Balanced distribution of workload along with the minimal overhead due to communication/synchronization is indispensable to reduce the parallel execution time. In the next sub-section, we describe the workload distribution that allows us to obtain a completely balanced workload distribution without any communication/synchronization overhead. Then, we present the parallel algorithm for schedulability analysis and finally characterize its theoretical performance.

Workload Distribution

We develop policies to distribute elements of each testing set $\Upsilon^{\mathsf{SC}_Z(i,j,\phi)}$ among the processors for the parallel algorithm. Our approach emphasizes a balanced distribution of workload among the processors to obtain a near-ideal speedup. Since the workload for the parallel algorithm is entirely dependent upon the testing sets for the schedulability conditions of Equation 9.2, a naive approach is to assign elements of each testing set $\Upsilon^{\mathsf{SC}_Z(i,j,\phi)}$ in a round-robin fashion to each of the processors; i.e., processor P_1 would test schedulability condition SC_Z for the first testing-set element, processor P_2 would test SC_Z for the second testing-set element, and so on. In general, after evaluating a testing set element t_i, processor P_k will skip the next m testing set elements which implies that each processor will work with a single element among the m consecutive members of $\Upsilon^{\mathsf{SC}_Z(i,j,\phi)}$. Thus, the total number of elements to be checked by each processor is at least $\left\lfloor \frac{|\Upsilon^{\mathsf{SC}_Z(i,j,\phi)}|}{m} \right\rfloor$ and the maximum difference in workload between two successive processors is one testing element. Now consider the next testing set; again, if the first processor P_1 tests the first element, then, in the worst case this processor may receive one more testing set element than the other processors. Therefore, at the end of checking condition $\mathsf{SC}_Z(i,j,\phi)$ of Equation 9.2, there may be a difference in workload of one testing element among processors. At the end of the execution, this difference could be equal to the total number of testing sets for all five conditions (which is $\mathcal{C}q^2$). This uneven workload will reduce the speedup of the parallel algorithm. Thus, in our approach we do not always allow the first processor P_1 to test the first element of the testing set, rather we keep track of the processor P_k that tests the last element in previous testing set. Then, we allow the next processor P_{k+1} to test the first element in the current testing set. To support the equal distribution of testing set elements, each processor maintains a root distribution variable r_k, where $r_k \in \{1, 2, \ldots, q\}$. The variable r_k indicates the starting element for P_k for the next testing set. The set below represents the subset of elements of $\Upsilon^{\mathsf{SC}_Z(i,j,\phi)}$ for which P_k is responsible for testing the condition SC_Z while P_k's root variable is r_k.

$$\Upsilon^{\mathsf{SC}_Z(i,j,\phi)}_{k,r_k} = \left\{x \middle| \left(x \in \Upsilon^{\mathsf{SC}_Z(i,j,\phi)}\right) \wedge \left(r_k \equiv x \mod m\right)\right\} \tag{9.3}$$

We now consider how to update the root variables to ensure a completely-balanced distribution of the testing set elements. For processor P_k with root variable r_k, we can determine the processor that has r_ℓ equal to one for some $\ell \in \{1, \ldots, m\}$. The expression $r'_k \stackrel{\text{def}}{=} ((k - r_k) \mod m) + 1$ identifies the rank of this processor. By distributing each of the elements of $\Upsilon^{\mathsf{SC}_Z(i,j,\phi)}$ in a round-robin fashion (as described in Equation 9.13), the first processor to receive an element has rank equal to $\ell' \stackrel{\text{def}}{=} \big(\big(r'_k + |\Upsilon^{\mathsf{SC}_Z(i,j,\phi)}| - 1\big) \mod m\big) + 1$. Rank ℓ' identifies the processor that will receive the first element in the next testing set distribution. Thus, for any other processor P_k to determine its new root variable, we must calculate $((k - \ell') \mod m) + 1$. Thus, we must use the following update rule:

$$r_k = \Big(\Big(k - \big(\big(r'_k + |\Upsilon^{\mathsf{SC}_Z(i,j,\phi)}| - 1\big) \mod m\big) + 1\Big) \mod m\Big) + 1. \tag{9.4}$$

After distributing the entire workload of all testing sets according to this rule, the difference between any two processors with respect to the number of testing set elements assigned is at most one. In addition to a completely-balanced workload distribution, we observe that the testing set elements do not need to be distributed (via communication or initialization) to the processors. In fact, since the testing set simply consists of consecutive integers, each processor independently generates testing set elements as needed, according to the set defined in Equation 9.13. Thus, the proposed distribution eliminates the communication overhead due to the workload distribution.

Algorithm 4 SUBI-PAR($\mathcal{M}$)

```
{Processor $k$ executes:}
$r_k \leftarrow Initialize()$
for $i = 1$ to $q$ do
    if CheckConditions-PAR$(r_k, 1, i, \emptyset, \emptyset, \emptyset)$ = false then
        return false
    end if
    for $j = 1$ to $q$ $(j \neq i)$ do
        if CheckConditions-PAR$(r_k, 3, i, j, \emptyset, \emptyset)$ = false then
            return false
        end if
        for $s = 0$ to $\delta_{ij}$ do
            if CheckConditions-PAR$(r_k, 2, i, j, s, \emptyset)$ = false then
                return false
            end if
        end for
    end for
end for
$\zeta \leftarrow$ MaxCarry-PAR$(\mathcal{M}, k, r_k)$
for $i = 1$ to $q$ do
    $C_i \leftarrow \max_{\{h=1,\dots,q\} \wedge h \neq i}\{\zeta_{hi}\}$
    for $j = 1$ to $q$ $(j \neq i)$ do
        for $\phi = 0$ to $\delta_{ij}$ do
            if CheckConditions-PAR$(r_k, 4, i, j, \phi, C_i)$ = false then
                return false
            end if
        end for
        for all $t = 0$ to $\mathcal{T}_{ij}$ do
            if CheckConditions-PAR$(r_k, 5, i, j, t, C_i)$ = false then
                return false
            end if
        end for
    end for
end for
return true
```

Algorithm Description

We now present the pseudocode for SUBI-PAR, our proposed parallel algorithm for schedulability analysis, in Algorithm 4. The algorithm is designed to run concurrently on all available processors. The SUBI-PAR uses two subroutines CheckConditions-PAR and MaxCarry-PAR. The algorithm starts with the initialization of the parallel execution. The rank of the processor (denoted by k) and the total number of processors are determined at this point. A data distribution root r_k, associated with each processor P_k, is initialized to the unique rank k of the processing platform. The algorithm then starts checking each testing set $\Upsilon^{\mathsf{SC}_Z(i,j,\phi)}$ of condition $\mathsf{SC}_Z(i,j,\phi)$ for all legal values of i, j and Z using the function CheckConditions-PAR$(r_k, Z, i, j, \phi, C_i)$ at each processor P_k. If the function CheckConditions-PAR returns true for the current testing set, the function will continue its execution to the next testing set; otherwise, the algorithm returns false; that is, the multi-mode real-time system $\mathcal{M}$ is not schedulable.

The evaluation of inequalities related to condition SC_1 of Equation 9.2 is performed in Lines 4 to 6. The condition SC_1 is evaluated for each of the q different modes. All remaining four conditions of Equation 9.2 are defined for pairs of modes; therefore, we use two nested *for*-loops to iterate through the testing sets associated with each such pair of modes. However, we separate the code segment related to conditions SC_4 and SC_5 (Lines 19 to 33) from the rest as the former two conditions require pre-computed carry-in executions calculated by the MaxCarry-PAR function (Line 6). The function MaxCarry-PAR could potentially be invoked at the beginning of the algorithm; in that case, all five conditions could be evaluated using one single block of nested loop. However, the function MaxCarry-PAR is a costly operation, and we allow its execution only if it is required. For unschedulable systems, it may be the case that the system will not satisfy one of the first three conditions: SC_1, SC_2, or SC_3; therefore, there is no need of invoking the costly MaxCarry-PAR for such unschedulable systems.

The algorithm CheckConditions-PAR is a case-based implementation for evaluating each condition $\mathsf{SC}_Z(i,j,\phi)$ using the condition variable Z. Depending on the value of $Z \in \{1,2,3,4,5\}$, the function selects the appropriate schedulability condition for each testing set element $x \in \Upsilon_{k,r_k}^{\mathsf{SC}_Z(i,j,\phi)}$. The per-processor testing set $\Upsilon_{k,r_k}^{\mathsf{SC}_Z(i,j,\phi)}$ is decided by its current data-distribution root r_k. As shown in Equation 9.13, this is an ordered set of evenly separated (of size m) positive integers starting from r_k; therefore, we allow each processor to generate its dataset associated with each inequality to reduce the overhead related to data distribution. After validating

Algorithm 5 CheckConditions-PAR$(r_k, Z, i, j, \phi, C_i)$.

```
1: result $\leftarrow$ true
2: for all $x$ in $\Upsilon_{k,r_k}^{\mathsf{sc}_Z(i,j,\phi)}$ do
3:    if $Z = 1$ then
4:       if $\mathsf{sbf}(\Omega^{(i)}, x) < \mathsf{dbf}(\tau^{(i)}, x)$ then
5:          result $\leftarrow$ false; break;
6:       end if
7:    else if $Z = 2$ then
8:       if $\mathsf{dbf}(\tau^{(j)} \setminus \tau^{(ij)}, x) + \mathsf{dbf}(\tau^{(ij)}, x + \phi) > \beta_{\mathsf{post}}^{i,j}(\phi, x)$ then
9:          result $\leftarrow$ false; break;
10:      end if
11:   else if $Z = 3$ then
12:      if $\mathsf{dbf}(\tau^{(ij)}, x) > \mathsf{sbf}(\Omega^{(ij)}, x)$ then
13:         result $\leftarrow$ false; break;
14:      end if
15:   else if $Z = 4$ then
16:      if $\Psi_x(M^{(i)}, M^{(j)}, C_i, \phi) > \beta_{\mathsf{trans}}^{i,j}(\phi)$ then
17:         result $\leftarrow$ false; break;
18:      end if
19:   else if $Z = 5$ then
20:      carry $\leftarrow \left(\Psi_x(M^{(i)}, M^{(j)}, C_i, \delta_{ij} + \phi) - \beta_{\mathsf{trans}}^{i,j}(\delta_{ij})\right)_+$
21:      if carry $+ \mathsf{dbf}(\tau^{(i)} \setminus \tau^{(ij)}, \phi) > \beta_{\mathsf{post}}^{i,j}(0, \phi)$ then
22:         result $\leftarrow$ false; break;
23:      end if
24:   end if
25: end for
26: Update $r_k$ using Equation 9.15.
27: return All-to-All-Reduce(result, AND)
```

the testing set, the data distribution root r_k at the processor P_k is updated using Equation 9.15. The function CheckConditions-PAR synchronizes schedulability results with all other executing processors using an All-to-All-Reduce operation with AND as the reduction operator. The CheckConditions-PAR returns false even if there is a single violation.

The function MaxCarry-PAR evaluates the sequence $\mathsf{ci}_0^{M^{(i)},M^{(j)}}$, $\mathsf{ci}_1^{M^{(i)},M^{(j)}}$, $\mathsf{ci}_2^{M^{(i)},M^{(j)}}$, ... using a *repeat-until* loop to calculate the carry-in $\mathsf{ci}(M^{(i)}, M^{(j)})$ for all pairs of modes and stores all the carry-in executions in a $q \times q$-matrix ζ. For all pairs, we calculate $\mathsf{ci}_g^{M^{(i)},M^{(j)}}$ at each step g from the value calculated at $(g-1)$-th step, and store the value in ζ_{ij} only if the new value is greater than the previous one. The function marks the change by setting the change flag to true. The newly calculated matrix ζ is synchronized using a All-to-All-Reduce operation with a MAX operator for each individual q^2 cell items. The function proceeds to the next step if there is a change in previously calculated carry-in executions (the change is true). The algorithm uses a All-to-All-Reduce operation with an OR operator to determine whether the change is set to true by at least one processor. The function proceeds to next step only if the change has a true value after the synchronization. Otherwise, the function returns with current values stored in $q \times q$-matrix ζ.

Finally, if the execution of the algorithm reaches Line 34 of SUBI-PAR, we may safely declare that a multi-mode system $\mathcal{M}$ is EDF-schedulable.

9.3.3 Parallel Performance

We investigate the asymptotic performance of our proposed parallel algorithm by using well known parallel performance metrics which include *parallel execution time*, *speedup*, *efficiency*, and *cost* [39]. The parallel execution time, denoted by T_m, is the time elapsed between the start and the end of a parallel computation. The value of T_m depends on the actual workload, number of processors m, and the parallel overhead. T_m decreases at a slower rate as m increases. However, the parallel overhead also increases with m; therefore, after certain value of m, T_m may not experience a noticeable decrease. T_m for our algorithm is given by

$$T_m = O\left(\frac{\mathcal{C}^2 q^2 n}{m} + \mathcal{C} q^2 m + n\right) \tag{9.5}$$

The first term corresponds to the actual amount of work performed by each of the processors; it is obtained by dividing the total serial workload by the number of processors. The second term represents the overhead due to communication. The communication operation used in our algorithm is the All-to-All reduction among m processors which has a complexity of $O(mk)$, where k is the size of the message on which reduction is performed [39]. Since the reduction operation is invoked $O(\mathcal{C})$ times with a message size of q^2 (line of Algorithm 10) the overhead

Algorithm 6 $\mathsf{MaxCarry\text{-}PAR}(\mathcal{M}, k, r_k)$.

```
1: {Returns a $[q \times q]$ matrix $\zeta$.}
2: $\zeta \leftarrow 0$
3: Compute $E_{ij}$ using Equation 7.21.
4: **repeat**
5:     $\mathsf{change} \leftarrow \text{false}$
6:     **for** $i = 1$ to $q$ **do**
7:         $\mathsf{ci}^{(i)}_{\max} \leftarrow \max_{h=1,\ldots,q \wedge h \neq i} \{\zeta_{hi}\}$
8:         **for** $j = 1$ to $q$ **do**
9:             $c \leftarrow 0$
10:            $d \leftarrow \max(\delta_{ij} + d^{(ij)}_{max}, d^{(i)}_{max})$
11:            **for all** $x$ in $\Upsilon^{\mathsf{SC}_4(i,j,\phi)}_{k,r_k}$ **do**
12:                $c \leftarrow \max\left(c, \Psi_x(M^{(i)}, M^{(j)}, \mathsf{ci}^{(i)}_{\max}, d)\right)$
13:            **end for**
14:            Update $r_k$ using Equation 9.15.
15:            $c \leftarrow c - \beta^{i,j}_{\mathsf{trans}}(\delta_{ij})$
16:                $- \sum_{\tau^{(ij)}_\ell \in \tau^{(ij)}} \mathsf{dbf}\left(\tau^{(ij)}_\ell, d - \delta_{ij} - p^{(ij)}_\ell\right)$
17:            **if** $\min(c, E_{ij}) > \zeta_{ij}$ **then**
18:                $\zeta_{ij} \leftarrow \min(c, E_{ij})$
19:                $\mathsf{change} \leftarrow \text{true}$
20:            **end if**
21:        **end for**
22:    **end for**
23:    All-to-All-Reduce($\mathsf{change}$, OR)
24:    All-to-All-Reduce($\zeta$, MAX)
25: **until** $\mathsf{change} = \text{false}$
26: **return** $\zeta$
```

due to communication is given by $O(\mathcal{C}q^2m)$. The third term represents the overhead due to workload imbalances; per the discussion after Equation 9.15, the difference, between any two processors, in the number of testing set elements is at most one. This testing set element requires $O(n)$ to evaluate any SC_Z.

The speedup S is defined as the ratio of the serial execution time T_s of the best sequential algorithm to the parallel execution time. That is, $S \overset{\text{def}}{=} \frac{T_s}{T_m}$. The perfect speedup for a parallel algorithm equals to m which may be difficult to achieve for algorithms that require communication/synchronization for the correct operation. Due to overhead, S may decrease as m increases. The speedup our algorithm is given by:

$$S \approx \frac{\mathcal{C}^2 q^2 n}{\frac{\mathcal{C}^2 q^2 n}{m} + \mathcal{C} q^2 m + n} \tag{9.6}$$

Efficiency, denoted by E, is a measure of the fraction of time for which the processors are usefully employed in solving the problem; it is defined as the ratio of the speedup to the number of processors, $E \overset{\text{def}}{=} \frac{S}{m}$. The efficiency accounts for the parallel overhead, and usually decreases as m increases. The efficiency of our proposed algorithm is given by:

$$E \approx \frac{\mathcal{C}^2 q^2 n}{m(\frac{\mathcal{C}^2 q^2 n}{m} + \mathcal{C} q^2 m + n)} \tag{9.7}$$

As mentioned in the previous sections, the last two terms in the denominator of Equations 9.6 and 9.7 are due to the parallel overhead which increases with m. As long as the overhead is smaller than the time required to perform the actual computation, the parallel algorithm remains scalable. We now determine conditions to ensure the scalability of SUBI-PAR using the concept of cost and cost-optimality. The cost is the sum of the time that each processor spends solving the problem, including the time to perform the actual work and the overhead due to communication/synchronization. A parallel algorithm is cost-optimal [39] if the cost has the same growth as the execution time of the fastest known serial algorithm. The following theorem develops conditions to restrict the parallel overhead:

Theorem 9 SUBI-*PAR is cost-optimal if* $m = O(\sqrt{\mathcal{C}n})$.

Proof: By definition, the cost of a parallel algorithm is the product of the parallel execution time and the number of processors. The cost of our parallel algorithm is given by:

$$\mathsf{cost} = \mathcal{C}^2 q^2 n + \mathcal{C} q^2 m^2 + nm \tag{9.8}$$

For the problem considered in this thesis, the execution time of the fastest known serial algorithm is $O(\mathcal{C}^2 q^2 n)$. As the growth for the first term of Equation 9.8 is the same as the growth of the

Table 9.1: Experiment Setup

Parameter	Value
No. of Modes (q)	$\{8, 12, 16, 20\}$
n_i	8
Range of $p_\ell^{(i)}$	$[2500, 6000]$
Deadline $d_\ell^{(i)}$	$p_\ell^{(i)} \times 0.8$
Utilization $u^{(i)}$	0.45
$\Pi^{(i)}$	1250
$\Theta^{(i)}$	$1.5u^{(i)}\Pi^{(i)}$
$\Delta^{(i)}$	$\Pi^{(i)}$
δ_{ij}	1250
$N^{(i)}$	2
No. of Processors	$\{1, \ldots, 24\}$

execution time of the fastest serial algorithm, our parallel algorithm is cost-optimal if the second and the third term have the same growth as $O(\mathcal{C}^2q^2n)$; that is, $m^2 = O(\mathcal{C}n)$ (for the second term) and $m = O(\mathcal{C}^2q^2)$ (for the last term). A reasonable assumption of $n \leq \mathcal{C}$ holds since $\mathcal{C}$ is an upper bound on the execution of tasks and each task has an execution of at least one. Since $n\mathcal{C}$ grows slower than $\mathcal{C}^2q^2$, the algorithm is cost-optimal if the first condition is satisfied; that is $m = O(\sqrt{\mathcal{C}n})$ which implies that as long as m grows slower than $\sqrt{\mathcal{C}n}$, SUBI-PAR remains cost-optimal; thus, the theorem follows. ■

From the analysis in the previous paragraph, it is evident that whenever m grows slower than $\sqrt{\mathcal{C}n}$, the overhead of the SUBI-PAR algorithm is less than $O(\mathcal{C}^2q^2n)$. It may be also shown that the speedup of SUBI-PAR for a computationally large problem ($\mathcal{C} \gg n$) is close to m (near-perfect speedup) as the first term in the denominator of Equation 9.6 dominates for a larger $\mathcal{C}$.

9.3.4 Experimental Results

We perform experiments on a cluster of AMD Opteron computers which is part of the Wayne State University grid. Each computer has two 2.4GHz dual core processors and 4 or 16GB of RAM. The computers are connected through a Gbit Ethernet. We used MPICH-1.2.7 as the standard message passing interface. Value ranges for the parameters of the multi-modal system are listed in Table 9.1. We have previously established the efficacy of the schedulability analysis in [35] over the previous state-of-the-art (Phan et al. [63]); therefore, we measure the efficiency of SUBI-PAR in this chapter.

For the simulation, we generate a set of 12 tasks from the parameters described in the Ta-

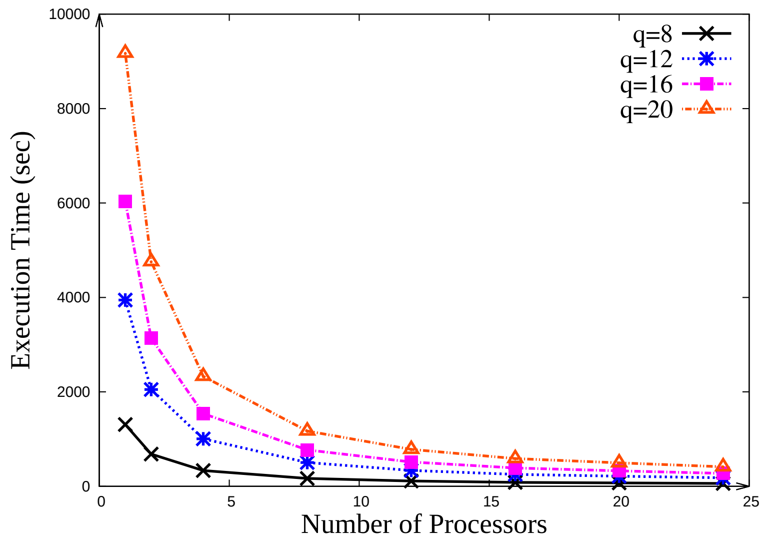

Figure 9.2: Execution time vs. number of processors.

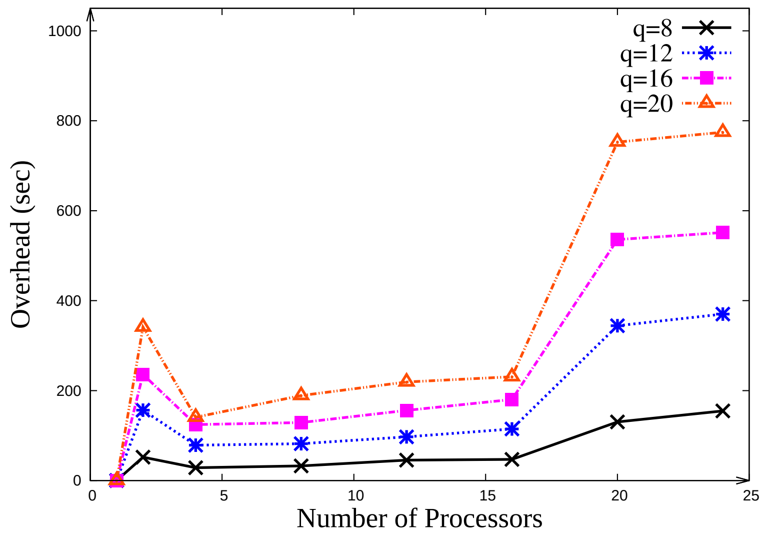

Figure 9.3: Parallel overhead vs. number of processors.

ble 9.1. Of the generated tasks, three are unchanged tasks and two are aborted tasks. We select at least eight tasks from the set for each mode $M^{(i)}$. The resource parameter of a mode is set based on the parameters described in the Table 9.1. In order to check the performance, we considered multi-mode systems with varying number of modes $q \in \{8, 12, 16, 20\}$. The SUBI-PAR algorithm, for each multi-mode system, is executed at least five times to reduce the effect on the execution time due to interference from other jobs in the grid. Among them, we took the minimum execution time for each multi-mode system. While checking the schedulability, the SUBI-PAR algorithm uses a total number of processors from the range $[1, 24]$. In Figure 9.2, we present the execution time of the SUBI-PAR algorithm for parallel systems with various numbers of processors. In this figure, the horizontal axis represents the total number of processors m while the vertical axis represents the execution time. Clearly, SUBI-PAR requires a smaller execution time for a larger number of processors. Note that the decrease in the execution time with the higher number of processors is not linear. This is due to the parallel overhead of our algorithms.

To calculate the overhead of the parallel execution, we consider the algorithm called SUBI (Schedulability Using Bounded Iteration) developed in the previous chapter as the best known serial algorithm for the problem. We execute this algorithm for each multi-mode system using the same hardware resource setting in the grid. The overhead T_o is calculated using the formula $mT_m - T_s$, where T_s is the execution time of SUBI. Figure 9.3 shows the overhead versus the number of processors used. Like most parallel algorithms, the parallel overhead of our proposed algorithm increases with q due to the increased communication/synchronization cost. However, the overhead does not obtain a noticeable increase after a certain limit on m. This limit depends on the number of modes (e.g., for $q = 8$, the limit is 20).

Figure 9.4 and Figure 9.5 show the parallel performance metrics for the SUBI-PAR. The speedup is calculated with respect to the execution time of SUBI. Figure 9.4 shows the speedup with respect to the number of processors. The speedup is close to the number of processors. Although it is not discernible in Figure 9.4, the speedup is slightly better for a larger number of modes. In Figure 9.5, we present the efficiency of the SUBI-PAR with respect to the total number of processors used. The parallel efficiency is calculated from the speedup and the number of

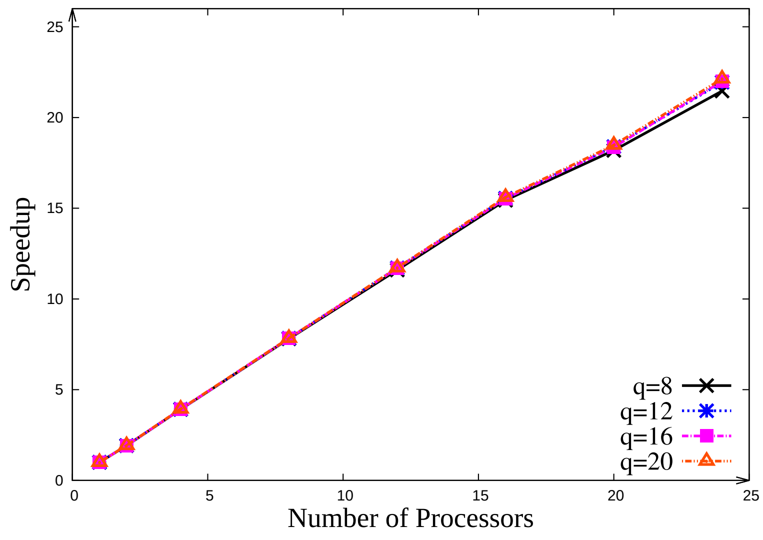

Figure 9.4: Speedup vs. number of processors.

processors used. The efficiency varies between $90 - 98\%$ in our experiments. Like the speedup factor, efficiency varies with the number of modes and parameters associated with each mode. We obtain better efficiency for higher number of modes. One possible explanation could be the amount of workload to share among processors which increases with the number of modes.

In Figures 9.4 and 9.5, there are spikes at $m = 2$ due to the higher interference from other

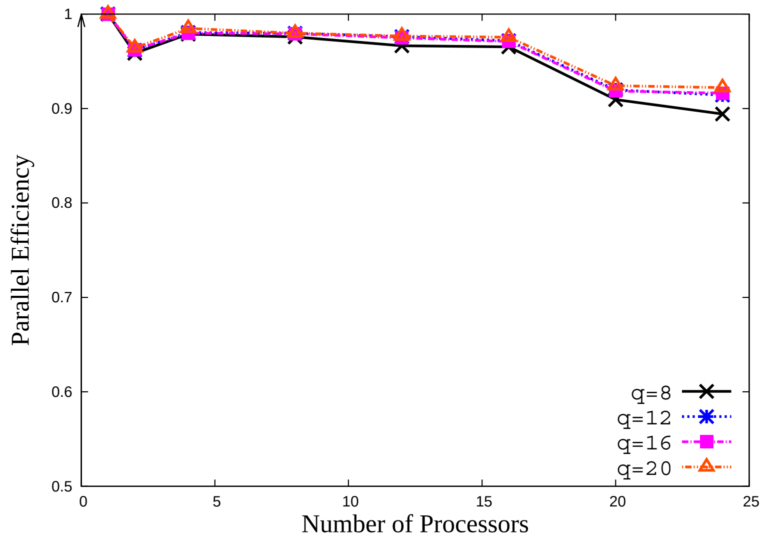

Figure 9.5: Efficiency vs. number of processors.

running jobs in grid. Each computer node in the grid has four cores, and the grid job scheduler assigns a single core for each processor requested unless explicitly specified. For $m = 2$, only two cores of a computer node are used by SUBI-PAR, and the remaining two cores may be utilized by other running jobs in the grid. Cores in the same node share memory and cache; therefore, the interference from outside jobs is higher for $m = 2$ than for m being a multiple of four where each node is occupied only by our schedulability test during its execution.

So far, we developed an algorithm for the parallel schedulability analysis of real-time systems with multiple hardware and software modes. The proposed parallel schedulability test is designed such that the overhead associated with the parallel execution is minimized to obtain better speedup/efficiency. The experimental results substantiate the efficacy of the proposed algorithm for parallel schedulability analysis; therefore, the algorithm can be used as an effective tool for the exploration of design-space while searching for optimal parameters of a multi-mode real-time system.

9.4 Parallel Schedulability Using GPU

To accelerate the schedulability analysis, in the previous section, we developed a parallel algorithm using message passing interface (MPI) to check the invariants [35] of the schedulable real-time multi-modal systems. Although this parallel algorithm greatly improves the total execution time, scaling essentially requires adding potentially expensive computational nodes. Today's massively parallel general-purpose Graphical Processing Unit (GPU) platforms are often a more cost effective alternative to scaling the number of general-purpose computer nodes. As GPUs are increasingly common for handheld devices, an efficient GPU-based schedulability analysis could also be used online to reconfigure the system by re-evaluating system schedulability if mode parameters change dynamically.

GPU architectures are being increasingly used as parallel processing platforms for solving large scale problems suitable for the SIMD (Single Instruction Multiple Data) execution model [39]. The scheduler of the GPU utilizes many threads, organized into warps [58, 57], that execute the same set of instructions with different data. As a result, branching in the currently executing instruction set of a warp and synchronization among different warps may affect the projected performance. In addition, synchronization requirements among GPU blocks may create possible deadlock scenarios [59] which distinguishes the GPU from many other parallel processing platforms. For load balancing upon a GPU platform, different centralized [39] and distributed techniques [22] (e.g., sender/receiver-initiated [31]) may not be suitable due to added overhead. We propose a decentralized load-balancing technique that reduces the overhead of data distribution. Experimental results establish that the speedup of our GPU-based algorithm is greater than the previous parallel implementation [5].

9.4.1 GPU-Based Schedulability

An important step in designing parallel algorithms is to decide the workload distribution among available processors. Workload distribution must account for the properties of the underlying processing platforms in order to achieve higher speedup. In this section, we first describe the GPU platform and then develop policies that take into account advantages and limitations of the GPU platform while distributing testing sets among processing elements. We emphasize a balanced workload distribution to decrease the overhead due to communication/synchronization and thus reduce the execution time.

GPU Platform

The CPU directs image processing tasks to the GPU, which relies heavily on arithmetic and logical operations, where image data is sent as a stream through a hardware graphics pipeline. This pipeline renders a stream in separate parts to construct an image. A GPU device has streaming multiprocessors (SM) each of which contains a fixed set of processing cores. This streaming architecture executes single instruction multiple data (SIMD) in parallel where each SM is computationally independent from any other SM, making it ideal for problems requiring large data sets processing. The Compute Unified Device Architecture (CUDA) [59] provides the API to submit tasks to and receive results from the graphics processor. The computations are performed by calling a method from the CPU that hosts the GPU device known as a *kernel* function. Processing threads are created and grouped together in *blocks*. The number of threads and blocks are parameters of the kernel function. A block is executed by the GPU scheduler in subsets of 32 parallel threads (known as a warp). Each block and thread have a unique index during the kernel execution. The platform maintains the built-in variables `blockIdx` and `threadIdx` to identify these indices.

Any structure placed on the GPU global memory can be accessed by the GPU threads and the CPU. Shared memory structures are local to each streaming multiprocessor with access restricted to a block of threads. Maximizing the number of accesses to shared memory rather than the global memory is an important optimization since access to the shared memory can be 100 times faster than the global memory. This approach needs to be balanced with the amount of overhead spent on transferring information back and forth between the GPU and CPU memory structures. The cost effective performance of the GPU and ease of use by extending an API based on the C programming language are strong contributors to the success of the CUDA architecture. Ease of use is attributed to the GPU scheduler. The scheduler automatically manages the execution of threads with some explicit synchronization. Thousands of threads can be scheduled efficiently, taking advantage of the available parallelism. Careful tuning of design parameters such as the amount of shared memory used, the number of threads, and the number of blocks can result in significant performance gains.

In order to efficiently utilize the full computational power of a GPU, we evaluate each of the five schedulability conditions using a *group* of blocks $\{\mathcal{P}^0, \ldots, \mathcal{P}^{G-1}\}$, where G is the number of groups on the GPU. The number of blocks in each $\mathcal{P}^g$ is denoted by B. We invoke a GPU kernel for each condition SC_Z once. Our policy is to evaluate each testing set by a single group $\mathcal{P}^g$, which implies that no two groups evaluate the same testing set. Since the size of testing sets varies with each condition SC_Z, the system designer may change B when evaluating different SC_Z to obtain similar execution times for all testing sets. The number of testing sets also varies significantly with SC_Z. Therefore, the system designer may utilize different G with different

SC_Z, but we restrict B and G to be fixed during the evaluation of a single SC_Z (i.e., throughout the execution of a single kernel invocation). The blocks per group B must be an integer and each group should contain at least one block (i.e., $B \geq 1$). Without loss of generality, we fix the number of threads per block to T. So we denote each group by $\mathcal{P}^g \overset{\text{def}}{=} \{\mathcal{P}_0^g, \mathcal{P}_1^g \ldots, \mathcal{P}_{B-1}^g\}$, where $\mathcal{P}_\beta^g$ is a block composed of threads $\{\mathcal{P}_{\beta,0}^g \ldots \mathcal{P}_{\beta,T-1}^g\}$.

Testing Set Distribution

In Section 5.1, we denoted by $\Upsilon^{\mathsf{SC}_Z(i,j,\phi)}$ the individual testing set of the condition SC_Z for the pair $(M^{(i)}, M^{(j)})$ at an interval of length ϕ. We use the notation Υ^{SC_Z} and $\Upsilon^{\mathsf{SC}_Z(i,j)}$ respectively to denote the set of all testing sets associated with the condition SC_Z and all testing sets of the pair $(M^{(i)}, M^{(j)})$ for condition SC_Z. These notations are defined as follows

$$\Upsilon^{\mathsf{SC}_Z(i,j)} = \left\{\Upsilon^{\mathsf{SC}_Z(i,j,\phi)} | 0 < \phi \leq \mathrm{TS}_Z(i,j)\right\} \tag{9.9}$$

and

$$\Upsilon^{\mathsf{SC}_Z} = \left\{\Upsilon^{\mathsf{SC}_Z(i,j)} | (0 \leq i,j < q) \land (i \neq j)\right\}. \tag{9.10}$$

Each testing set of Υ^{SC_Z} can be uniquely identified by i, j, and ϕ. To obtain a unique index

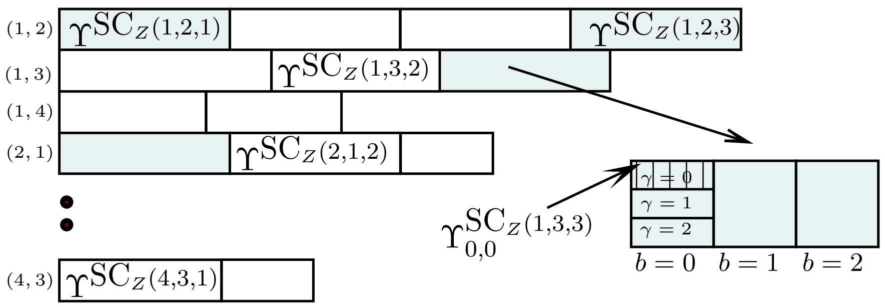
(1, 2)
$\Upsilon^{\mathrm{SC}_Z(1,2,1)}$
$\Upsilon^{\mathrm{SC}_Z(1,2,3)}$
(1, 3)
$\Upsilon^{\mathrm{SC}_Z(1,3,2)}$
(1, 4)
(2, 1)
$\Upsilon^{\mathrm{SC}_Z(2,1,2)}$
(4, 3)
$\Upsilon^{\mathrm{SC}_Z(4,3,1)}$
$\Upsilon_{0,0}^{\mathrm{SC}_Z(1,3,3)}$
$\gamma = 0$
$\gamma = 1$
$\gamma = 2$
$b = 0$
$b = 1$
$b = 2$

Figure 9.6: Parallel workload distribution for checking schedulability. G, B and T are assumed to be 3. Each row represents testing sets of $\Upsilon^{\mathsf{SC}_Z(i,j)}$ (associated with mode pair $M^{(i)}$ and $M^{(j)}$) where each rectangle denotes an individual testing set $\Upsilon^{\mathsf{SC}_Z(i,j,\phi)}$. Shaded rectangles belong to $\Upsilon_0^{\mathsf{SC}_Z}$ which is assigned to the group 0. The bottom rectangle at the right depicts the distribution of the testing set $\Upsilon^{\mathsf{SC}_Z(1,3,3)}$ among all threads of the group 0.

for each testing set $\Upsilon^{\mathsf{SC}_Z(i,j,\phi)}$, we define the following function:

$$\mathcal{X}_Z(i,j,\phi) = \sum_{x=1}^{i-1} \sum_{\substack{1\le y\le q \\ x\neq y}}^{q} \mathrm{TS}_Z(x,y) + \sum_{\substack{y=1 \\ i\neq y}}^{j-1} \mathrm{TS}_Z(i,y) + \phi. \tag{9.11}$$

We distribute all testing sets among the available groups (recall that each testing set is executed by a single group). A testing set $\Upsilon^{\mathsf{SC}_Z(i,j,\phi)} \in \Upsilon^{\mathsf{SC}_Z}$ will be assigned to a particular group based on the index function $\mathcal{X}_Z(i,j,\phi)$. The testing sets corresponding to a group $\mathcal{P}^g$ can be determined as follows:

$$\Upsilon_g^{\mathsf{SC}_Z} = \left\{ \Upsilon^{\mathsf{SC}_Z(i,j,\phi)} \middle| \, (g \equiv \mathcal{X}_Z(i,j,\phi) \mod G) \right\}. \tag{9.12}$$

The elements of $\Upsilon^{\mathsf{SC}_Z(i,j,\phi)} \in \Upsilon_g^{\mathsf{SC}_Z}$ are distributed among the blocks of a group $\mathcal{P}^g$. Each testing set is divided into partitions of size $\left\lceil \frac{\Upsilon^{\mathsf{SC}_Z(i,j,\phi)}}{B} \right\rceil$ (the workload distribution is depicted in Figure 9.6). A block $\mathcal{P}^g_\beta$ is responsible for the β-th partition of that testing set and distributes the partition among its threads $\mathcal{P}^g_{\beta,\gamma} \in \mathcal{P}^g_\beta$. We emphasize a balanced distribution of partition elements among these threads to increase speedup. We consider a round-robin distribution [39] of a partition among threads of $\mathcal{P}^g_\beta$. We take into account the continuation of the last partition: that is, we start assigning the first element of a partition to a thread $\mathcal{P}^g_{\beta,\gamma}$ where the previous thread $\mathcal{P}^g_{\beta,\gamma-1}$ evaluates the last element of the previous partition. In order to achieve this, each thread $\mathcal{P}^g_{\beta,\gamma}$ maintains a root variable r where $r \in \{0,\dots,(T-1)\}$ (motivated by Ahmed et al. [5]). The variable r indicates the index of the first element assigned to $\mathcal{P}^g_{\beta,\gamma}$ from the next partition. The set below represents the partition of $\Upsilon^{\mathsf{SC}_Z(i,j,\phi)}$ assigned to a thread $\mathcal{P}^g_{\beta,\gamma}$ where $\mathcal{P}^g_{\beta,\gamma}$'s root variable is r.

$$\begin{aligned} \Upsilon_{\kappa,r}^{\mathsf{SC}_Z(i,j,\phi)} &= \Big\{ x \big| (x \in \Upsilon^{\mathsf{SC}_Z(i,j,\phi)}) \wedge (r \equiv x \mod T) \wedge \\ &\quad \Big(\beta \times \left\lceil \tfrac{\Upsilon^{\mathsf{SC}_Z(i,j,\phi)}}{B} \right\rceil \le x < (\beta+1) \left\lceil \tfrac{\Upsilon^{\mathsf{SC}_Z(i,j,\phi)}}{B} \right\rceil \Big) \Big\} \end{aligned} \tag{9.13}$$

Table 9.2: Notations in GPU Based Schedulability

Expression	Description
$\tau^{(i)}$	Real-time workload of $M^{(i)}$
$\Omega^{(i)}$	Minimum resource of $M^{(i)}$
$\tau_\ell^{(i)}$	The ℓ-th sporadic task of $\tau^{(i)}$
$e_\ell^{(i)}$	Execution requirement of $\tau_\ell^{(i)}$
$d_\ell^{(i)}$	Relative deadline of $\tau_\ell^{(i)}$
$p_\ell^{(i)}$	Period of $\tau_\ell^{(i)}$
$u_\ell^{(i)}$	Utilization of $\tau_\ell^{(i)}$
$\Pi^{(i)}$	Resource period for $\Omega^{(i)}$
$\Theta^{(i)}$	Resource capacity of $\Omega^{(i)}$
$\Delta^{(i)}$	Resource deadline of $\Omega^{(i)}$
G	Total number of groups
B	Total number of blocks in a group
T	Total number of threads per block
$\mathcal{P}^g$	The g-th group
$\mathcal{P}^g_\beta$	The β-th block of group $\mathcal{P}^g$
$\mathcal{P}^g_{\beta,\gamma}$	The γ-th thread of block $\mathcal{P}^g_\beta$
r	Data distribution root variable for $\mathcal{P}^g_{\beta,\gamma}$
$\Upsilon^{\mathsf{SC}_Z(i,j,\phi)}$	Testing sets for the pair $(M^{(i)}, M^{(j)})$ at ϕ
Υ^{SC_Z}	Set of testing sets for condition SC_Z
$\Upsilon^{\mathsf{SC}_Z}_g$	Testing sets of SC_Z for group g
$\Upsilon^{\mathsf{SC}_Z(i,j,\phi)}_{\psi(g,\beta,\gamma),r}$	Elements for which thread $\mathcal{P}^g_{\beta,\gamma}$ is responsible.

where $\kappa = \psi(g, \beta, \gamma)$ such that

$$\psi(g, \beta, \gamma) = g \times B \times T + \beta \times T + \gamma. \tag{9.14}$$

Note that the function $\psi(g, \beta, \gamma)$ gives a system-wide unique identifier to each thread $\mathcal{P}^g_{\beta,\gamma}$. For $\mathcal{P}^g_{\beta,\gamma}$ with root variable r, we determine $\mathcal{P}^g_{\beta,\rho}$ that has $r = 0$ for some $\rho \in \{0, \ldots, (T-1)\}$. The expression $X \overset{\text{def}}{=} ((\gamma - r) \mod T)$ identifies this thread. By distributing each of the elements of $\Upsilon^{\mathsf{SC}_Z(i,j,\phi)}_{k,r}$ in a round-robin fashion, the first thread to receive an element has the index equal to $\rho' \overset{\text{def}}{=} \left(\left(X + \left\lceil \frac{\Upsilon^{\mathsf{SC}_Z(i,j,\phi)}}{B} \right\rceil\right) \mod T\right)$. To ensure a load balanced distribution, ρ' identifies the

Algorithm 7 SUBI-GPU$(\mathcal{M}, B)$

```
1:  {CPU executes:}
2:  $G \leftarrow q$
3:  proceed $\leftarrow \mathrm{true}$
4:  for $Z = 1$ to $5$ do
5:      if $Z = 4$ then
6:          $\zeta \leftarrow$ MaxCarry-GPU$(\mathcal{M}, B)$
7:      end if
8:      Invoke CheckConditions-GPU$(Z, \zeta, B)$ with $G$ groups.
9:      Perform AND on returned values from line 8 to set proceed.
10:     if proceed $= \mathrm{false}$ then
11:         break;
12:     end if
13:     $G \leftarrow q(q-1)$
14: end for
15: return proceed
```

thread that will receive the first element in the next partition. Thus, for any other thread $\mathcal{P}^{g}_{\beta,\gamma}$ to determine its new root variable, we must calculate $((\gamma - \rho') \mod T)$. The update rule for r after the execution of a partition at each thread is

$$r = \left(\gamma - \left(\left(X + \left\lceil \frac{\Upsilon^{\mathsf{SC}_Z(i,j,\phi)}}{B} \right\rceil\right) \mod T\right)\right) \mod T. \tag{9.15}$$

9.4.2 Algorithms

In this section, we develop a parallel algorithm for checking schedulability of a real-time multi-modal system using policies defined in the previous section. The main algorithm is SUBI-GPU, which utilizes three subroutines to perform the schedulability analysis. Conditions SC_4 and SC_5 require the maximum carry-in for all pairs of modes, which are calculated by MaxCarry-GPU. Table 9.2 lists the notations to describe the algorithms.

SUBI-GPU Algorithm

The pseudo-code for SUBI-GPU is given in Algorithm 7, which is designed to execute in the CPU. SUBI-GPU invokes CheckConditions-GPU for each of the five conditions with varying numbers of groups denoted by G, where $G = q$ if $Z = 1$, and $G = q(q-1)$ otherwise. As discussed in the section for workload distribution, the number of testing sets and their sizes varies significantly over different conditions. For example, SC_1 requires only the evaluation of q testing sets whereas all other conditions require at least $q(q-1)$ testing sets (the condition SC_5 requires the largest number of testing sets, which is $\sum_{i,j \leq q} \mathrm{TS}_5(i,j)$). Therefore, the system designer may want to use a varying number of groups for different SC_Z as a higher number of groups may increase the parallel efficiency of a GPU platform; however, a total number of groups greater than the number of testing sets for any SC_Z may not be a good policy as all groups with a group identifier g greater than the number of testing sets will not find any work to complete; thus, these groups will increase the overhead.

CheckConditions-GPU Subroutine

The pseudo-code for CheckConditions-GPU is given in Algorithm 8. CheckConditions-GPU is designed to execute in the GPU to check each condition SC_Z. The number of groups G is passed to CheckConditions-GPU as an argument, but the actual GPU block should be a multiple of B if $B > 1$. The built-in variables `blockIdx` and `threadIdx` are initialized by the runtime system and accessed within the kernel. A block's group index g equals $\left\lfloor \frac{\texttt{blockIdx}}{B} \right\rfloor$ and its block index β within a group equals ($\texttt{blockIdx} \mod \texttt{B}$). The thread index γ is initialized by the construct `threadIdx`. We create a function Init($\texttt{blockIdx}, \texttt{threadIdx}, \texttt{B}$) on line 2 that initializes (g, β, γ) using the aforementioned rules. The root variable r is initialized with the GPU thread identifier `threadIdx`. The *for*-loop in lines 4 to 33 iterates over each pair for all pairs of modes (optionally, this *for*-loop could be replaced by two cascaded *for*-loops iterating over $i, j < q$ and $i \neq j$). After identifying (i,j), the loop in lines 6 to 32 iterates over all testing sets assigned to group $\mathcal{P}^g$. For each $\Upsilon^{\mathsf{sc}_Z(i,j,\phi)}$, the thread $\mathcal{P}^g_{\beta,\gamma}$ considers its partition $\Upsilon^{\mathsf{sc}_Z(i,j,\phi)}_{k,r}$

Algorithm 8 CheckConditions-GPU(Z, ζ, B).

```
1:  {Thread $\mathcal{P}^{g}_{\beta,\gamma}$ executes:}
2:  $(g, \beta, \gamma) \leftarrow$ Init$(\texttt{blockIdx}, \texttt{threadIdx}, \texttt{B})$
3:  $r \leftarrow \texttt{threadIdx}$
4:  **for all** $(i, j)$ of $q(q-1)$ pairs **do**
5:      $C_i \leftarrow \max_{\{h=1,\ldots,q\} \wedge h \neq i}\{\zeta_{hi}\}$
6:      **for all** $\Upsilon^{\mathsf{sc}_Z(i,j,\phi)}$ in $\Upsilon^{\mathsf{sc}_Z}_{g}$ **do**
7:          **for all** $x$ in $\Upsilon^{\mathsf{sc}_Z(i,j,\phi)}_{\kappa,r}$ **do**
8:              **if** $Z = 1$ **then**
9:                  **if** $\mathsf{sbf}(\Omega^{(i)}, x) < \mathsf{dbf}(\tau^{(i)}, x)$ **then**
10:                     **return** false
11:                 **end if**
12:             **else if** $Z = 2$ **then**
13:                 **if** $\mathsf{dbf}(\tau^{(j)} \setminus \tau^{(ij)}, x) + \mathsf{dbf}(\tau^{(ij)}, x + \phi) > \beta^{i,j}_{\mathsf{post}}(\phi, x)$ **then**
14:                     **return** false
15:                 **end if**
16:             **else if** $Z = 3$ **then**
17:                 **if** $\mathsf{dbf}(\tau^{(ij)}, x) > \mathsf{sbf}(\Omega^{(ij)}, x)$ **then**
18:                     **return** false
19:                 **end if**
20:             **else if** $Z = 4$ **then**
21:                 **if** $\Psi_x(M^{(i)}, M^{(j)}, C_i, \phi) > \beta^{M^{(i)},M^{(j)}}_{\mathsf{trans}}(x)$ **then**
22:                     **return** false
23:                 **end if**
24:             **else if** $Z = 5$ **then**
25:                 carry $\leftarrow \Psi_x(M^{(i)}, M^{(j)}, C_i, \phi)$ - $\beta^{M^{(i)},M^{(j)}}_{\mathsf{trans}}(\delta^{(ij)})$
26:                 **if** carry $+ \mathsf{dbf}(\tau^{(i)} \setminus \tau^{(ij)}, x) > \beta^{i,j}_{\mathsf{post}}(0, x)$ **then**
27:                     **return** false
28:                 **end if**
29:             **end if**
30:         **end for**
31:         Update $r$ using Equation 9.15.
32:     **end for**
33: **end for**
34: **return** true
```

only where $k \overset{\text{def}}{=} \psi(g, \beta, \gamma)$ and iterates over each element x of this partition using the *for*-loop in lines 7 to 30. Based on Z, CheckConditions-GPU chooses the appropriate condition using a nested *if-then-else* block. Although a subroutine with branching does not perform well inside a

Algorithm 9 GetCarry-GPU$(\mathcal{M}, \zeta, B)$.

```
 1: {Each thread $\mathcal{P}^g_{\beta,\gamma}$ executes:}
 2: $(g, \beta, \gamma) \leftarrow$ Init$(\texttt{blockIdx}, \texttt{threadIdx}, \texttt{B})$
 3: change $\leftarrow$ false
 4: $i \leftarrow \left\lfloor \frac{g}{q-1} \right\rfloor$
 5: if $i < g \mod (q-1)$ then
 6:     $j \leftarrow g \mod (q-1)$
 7: else
 8:     $j \leftarrow g \mod (q-1) + 1$
 9: end if
10: Compute $E_{ij}$ using Equation 6.18.
11: $\mathsf{ci}^{(i)}_{\max} \leftarrow \max_{h=1,\ldots,q \wedge h \neq i} \{\zeta_{hi}\}$
12: $c \leftarrow 0$
13: $d \leftarrow \max(\delta^{(ij)} + d^{(ij)}_{max}, d^{(i)}_{max})$
14: for all $x$ in $\Upsilon^{\mathsf{SC}_4(i,j,\delta^{(ij)})}_{\kappa,r}$ do
15:     $c \leftarrow \max\left(c, \Psi_x(M^{(i)}, M^{(j)}, \mathsf{ci}^{(i)}_{\max}, d)\right)$
16: end for
17: $c \leftarrow c - \sum_{\tau^{(ij)}_\ell \in \tau^{(ij)}} \mathsf{dbf}\left(\tau^{(ij)}_\ell, d - \delta^{(ij)} - p^{(ij)}_\ell\right) - \beta^{M^{(i)},M^{(j)}}_{\mathsf{trans}}(\delta^{(ij)})$
18: if $\min(c, E_{ij}) > \zeta_{ij}$ then
19:     $\zeta_{ij} \leftarrow \min(c, E_{ij})$
20:     change $\leftarrow$ true
21: end if
22: return $(\zeta_{ij}, \text{change})$
```

GPU thread, we use an *if-then-else* form for ease of presentation. An actual implementation of CheckConditions-GPU can use five separate subroutines instead of nested *if-then-else*. The algorithm returns with false if SC_Z does not hold for any x. These returned values are collected by SUBI-GPU from the GPU global memory. After completing each partition, each thread updates its r (line 31) to obtain a balanced distribution.

We now develop algorithms that calculate the maximum carry-in for all pairs of modes. The maximum carry-in that a mode $M^{(i)}$ can forward to $M^{(j)}$, taking into account all possible previous mode-change requests, is given by $\mathsf{ci}(M^{(i)}, M^{(j)})$. The $\mathsf{ci}(M^{(i)}, M^{(j)})$ is calculated using the sequence $\mathsf{ci}_0^{M^{(i)},M^{(j)}}$, $\mathsf{ci}_1^{M^{(i)},M^{(j)}}$, $\mathsf{ci}_2^{M^{(i)},M^{(j)}}$, . . ., $\mathsf{ci}_\eta^{M^{(i)},M^{(j)}}$, ... In Chapter 6, we showed that the maximum value of η is limited by the total execution requirements of any mode which is

$\max_{1 \le i \le q} \sum_{1 \le \ell \le n_{(i)}} e_\ell^{(i)}$. The entire sequence may not be evaluated using a single kernel invocation of the GPU as the calculation of $\mathsf{ci}_\eta^{M^{(i)},M^{(j)}}$ requires the synchronization of results from all participating blocks. This calculation also depends on values calculated at the immediate previous step ($\mathsf{ci}_\eta^{M^{(h)},M^{(i)}}$ for all $h, i \le q$); therefore, an implementation of ci using a single kernel invocation may require inter-block synchronization. Xiao and Feng [82] addressed the inter-block synchronization which may occur at the end of a thread-execution; however, the same technique may not be suitable for the scenario where synchronization among threads of different blocks may occur inside a loop. These inter-block synchronizations may result in deadlock when all the GPU blocks are not executed concurrently by a GPU scheduler. We instead divide the calculation of ci into two procedures: $\mathsf{GetCarry}$-GPU and $\mathsf{MaxCarry}$-GPU. $\mathsf{MaxCarry}$-GPU is a CPU function that iterates over η and invokes the GPU function $\mathsf{GetCarry}$-GPU to calculate $\mathsf{ci}_\eta^{M^{(i)},M^{(j)}}$ where values calculated at the $(\eta - 1)$-th step is given as an argument.

$\mathsf{GetCarry}$-GPU Subroutine

The device subroutine $\mathsf{GetCarry}$-GPU is presented in Algorithm 9. Variables (g, β, γ) are initialized at the beginning by Init(`blockIdx`, `threadIdx`, `B`). Each group calculates carry-in for a single pair of modes. Lines 5 to 9 determine mode indices (i, j) from the group index g. For each pair of modes (i, j), there is a single testing set ($\Upsilon^{\mathsf{sc}_4(i,j,\phi)}$) to be evaluated. The block $\mathcal{P}_\beta^g$, in the group $\mathcal{P}^g$, is assigned a partition of consecutive testing set elements. Next, $\mathsf{GetCarry}$-GPU calculates $\mathsf{ci}_\eta^{M^{(i)},M^{(j)}}$ according to Equations 7.21 and 7.22; and stores the new value in ζ_{ij} if it

Algorithm 10 $\mathsf{MaxCarry}$-GPU$(\mathcal{M}, B)$.

```
1: {CPU executes:}
2: {Returns a $[q \times q]$ matrix $\zeta$.}
3: $\zeta \leftarrow 0$
4: **repeat**
5:     proceed $\leftarrow$ false.
6:     Invoke GetCarry-GPU$(\mathcal{M}, \zeta, B)$ using $q(q-1)$ groups.
7:     Perform OR on change from $q(q-1)$ groups to set proceed.
8: **until** proceed $=$ false
9: **return** $\zeta$
```

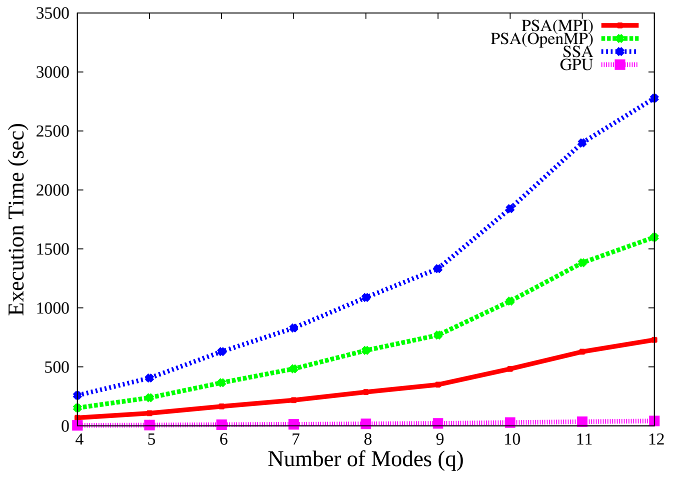

Figure 9.7: Execution time vs. number of modes (q)

is greater than the old value. The function marks the change by setting the change flag to true (lines 18 to 21).

MaxCarry-GPU Subroutine

The MaxCarry-GPU subroutine presented in Algorithm 10 calculates ci for all pairs of modes $M^{(i)}$ and $M^{(j)}$ using a *repeat-until* loop that invokes GetCarry-GPU with $q(q-1)$ groups at each iteration. GetCarry-GPU at each group calculates $\mathsf{ci}_{\eta}^{M^{(i)},M^{(j)}}$ for different pairs of modes from the value calculated at the previous step and stores the new value in ζ_{ij}. To determine proceed, MaxCarry-GPU performs an associative *or* operation (line 7) on all values of change returned from all groups. This step determines whether change is set to true by at least one thread. The function proceeds to the next step only if proceed is true. Otherwise, the function exits with the current values stored in ζ. The last step is analogous to All-to-All-Reduce with an OR operator [39].

9.4.3 Experimental Results

We compared SUBI with two existing algorithms: a serial schedulability analysis (denoted as SSA) by Fisher and Ahmed [35] and a parallel schedulability analysis (PSA) by Ahmed et al. [5]. PSA was executed upon a cluster of AMD Opteron computers on the Wayne State University grid. Each node has two 2.4GHz dual-core processors with 16GB of RAM. We have used two well-known parallel programming interfaces, MPI and OpenMP, to implement PSA. Both MPI and OpenMP implementations were executed on 2 processors (4 cores) in the cluster whereas SSA was executed on a single core from the same cluster. For SUBI, we used a GeForce GT 440 GPU upon a computer with a 2.33GHz Intel® Core™ 2 Duo processor and 2.0GB RAM. The GT 440 has two streaming multiprocessors (SM) each of which contains 48 cores. For generating the multi-modal systems, we have used the following parameters and value ranges:

1. The number of modes q of a multi-modal system is taken from the range $\{4, \ldots, 12\}$.

2. For the real-time workload of a multi-modal system, the UUniFast algorithm [16] is used to randomly generate a pool of 16 sporadic tasks by uniform distribution with total utilization 1. Each task period $p_\ell^{(i)}$ is uniformly drawn from $\{200, \ldots, 2500\}$ and $d_\ell^{(i)}$ is set to $p_\ell^{(i)}$.

3. For each mode $M^{(i)}$, 6 tasks have been chosen randomly from the pool of tasks using a uniform distribution.

We performed two sets of experiments. For the first set of experiments, we varied q from 4 to 12. We then measured the execution times for the SUBI algorithm upon the GPU platform, SSA upon a single node on the cluster, and both OpenMP and MPI implementations of PSA upon the

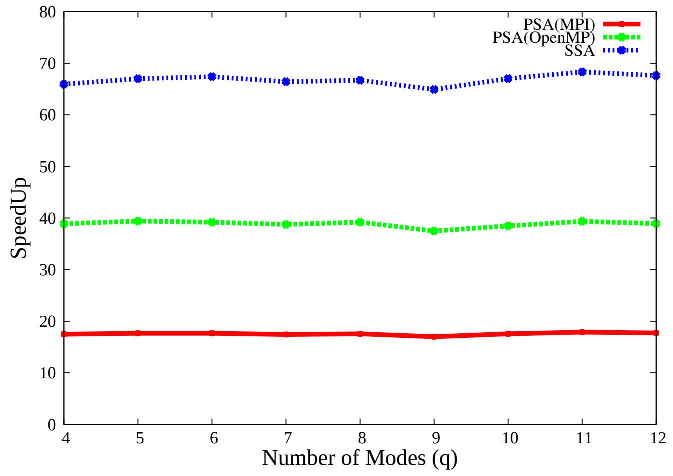

Figure 9.8: Speedup vs. number of modes (q)

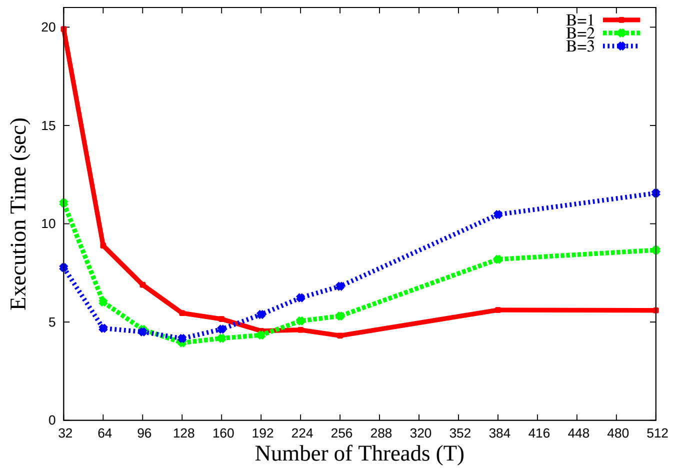

Figure 9.9: Performance of SUBI with varying blocks and threads.

cluster. For each q, we have generated 25 multi-modal systems based on the policy described in the previous section. In Figure 9.7, the horizontal axis represents the total number of modes q while the vertical axis represents the mean execution time associated with each q. Among the implementations, the MPI version of PSA performs better upon a computer cluster than the OpenMP implementation and SSA; however, the execution time (including data transfer time) of SUBI is significantly less than that of any PSA implementation.

9.5 Conclusion

In this chapter, we proposed a parallel schedulability analysis of real-time multi-modal systems that obtains significant speedup over existing schedulability analysis algorithms. The proposed algorithm takes advantage of favorable features (e.g., shared memory, parallel platform architecture) and avoids limitations (e.g., difficulty in achieving inter-block synchronization upon a GPU) of a parallel platform. The increased speedup will be especially beneficial for the process of determining the minimum resource parameters of a multi-modal system by repetitive application of a schedulability analysis with varying resources. This process is referred to as design-space exploration. We later extended the parallel schedulability analysis for a GPU platform. A fast schedulability analysis can further enhance the development of interactive tools (e.g., an intelligent personal assistant software for GPU enabled handheld devices) on top of real-time multi-modal systems.

CHAPTER 10: CONCLUSION & FUTURE WORK

Multi-modal systems is very effective to maintain stability in a dynamic and unpredictable environment. With the presence of dynamic power management (DPM) features even with low end processors, there will be more and more multi-modal real-time systems in future to exploit dynamic environment for guaranteed service. Researchers have been working on control computing [37, 36] systems to achieve this goal. To achieve control computing systems, multi-modal execution is the prerequisite. Each of the previously-proposed real-time control systems is *soft* real-time; that is, the system cannot guarantee that every deadline will be met, but is designed with the objective to minimize the number (or the effect) of deadline misses. We are unaware of a single feedback control computing system with hardware and software modes that is *hard* real-time, in that it guarantees that no deadline will be missed.

The current non-existence of such a control computing system is due to a fundamental gap in the research literature on effective and efficient multi-modal systems and schedulability analysis. This is due to the fact that real-time applications that meet all deadlines in the presence of changing execution modes is inherently difficult due to the challenge in predicting the aggregate computational resources that will be available to the real-time application over any interval of time. Therefore, all existing multimode schedulability analysis that handles both resource and application mode changes is highly exponential and not scalable for subsystems with a moderate or large number of modes. In this dissertation, we take an initial step towards the design of such a hard-real-time control system by providing a theoretical framework and associated time-efficient schedulability analysis.

The proposed multi-modal system and schedulability developed in this thesis was implemented by Hettiarachchi et al. [42]. The goal was to obtain performance guarantees in an unpredictable thermal environment. Towards this challenge, we have presented a control theoretic framework for thermal stress analysis in real-time systems. Our proposed method employs a nested feedback control system, which is based on optimum control theory. For our system, we derived strong guarantees for any real-time execution mode. Our method has the distinct advantage of being able to verify thermal aspects of a system before it is put into operation. In addition, we show via simulations that our framework performs as well as previous approaches which have no formal guarantee on the thermal. Our implementation upon a hardware testbed validates our proposed model and control framework. While related to the dissertation topic, the challenge of designing multi-modal control systems is not in the scope of research described in this dissertation. We refer the reader to papers by Hettiarachchi et al. [42] for details on multi-modal control system design. Following sections summarize the contribution of this dissertation.

10.1 Summary Results

10.1.1 Multi-Modal System

We proposed a model for hardware and software modes in a real-time system. We consider a system consisting of multiple subsystems, and exclusively focus on the subsystem-level schedulability. We consider that both the real-time application workload and the processing resource have multiple modes. In order to ensure schedulability along with temporal isolation, we couple each application mode with a processing resource to constitute a subsystem mode. Therefore, each subsystem mode is characterized by a pair: an explicit deadline periodic resource (i.e., hardware execution behavior) and a sporadic task system (i.e., software execution behavior).

10.1.2 EDF Schedulability Analysis

We derived EDF schedulability analysis for mode-change request for two settings: concrete and non-concrete mode change requests. For non-concrete systems (i.e., the sequence of mode-changes are not known a priori), we obtain a schedulability analysis algorithm that has pseudo-polynomial time complexity. The previous known algorithm which uses a reachability graph requires exponential time complexity. Our simulation results validate the effectiveness and efficiency of algorithm and demonstrate that it scales as the number of modes increases.

10.1.3 FP Schedulability Analysis

We present an efficient FP-schedulability analysis for our proposed multi-modal systems. In addition, our schedulability analysis for multi-modal systems can address non-preemptible execution of a task in a mode. Furthermore, we showed that our analysis can be done in tractable time complexity; therefore, this result may be used to calculate more refined (near optimal) resource parameters by repetitive application of this schedulability with varying hardware parameters as shown in Ahmed and Fisher [4].

10.1.4 Parallel Schedulability Analysis

We proposed a parallel algorithm for the EDF schedulability analysis of real-time systems with multiple hardware and software modes. The proposed parallel schedulability test is designed such that the overhead associated with the parallel execution is minimized to obtain better speedup or efficiency. The experimental results substantiate the efficacy of the proposed algorithm for parallel schedulability analysis; therefore, the algorithm can be used as an effective tool for the exploration of design space while searching for optimal parameters of a multimode real-time system.

We extended the parallel schedulability analysis for GPU computing that obtained significant speedup over general implementation (using MPI and OpenMP) of parallel schedulability analysis algorithms. The proposed GPU based algorithm takes advantage of favorable features (e.g., shared memory, SIMD architecture) and avoids limitations (e.g., difficulty in achieving inter-block synchronization) of a GPU platform. The increased speedup will be beneficial for determining the minimum resource parameters of a multi-modal system by repetitive application of a schedulability analysis with varying resources. This process is also referred as design-space exploration. A fast schedulability analysis can further enhance the development of interactive tools (e.g., intelligent personal assistant software for handheld devices) on top of real-time multi-modal systems.

10.1.5 Resource Estimation

Multi-modal systems developed so far have not addressed the optimal (hardware) resource allocation for each mode. By leveraging the developed schedulability analysis in the above subsections, we address the problem of minimizing a multimode real-time system with respect to resource usages over all modes. Optimal solutions may vary due to different objective functions (e.g., minimizing peak-temperature or minimizing total energy consumption). In [6], we have shown that optimizing the capacity of a periodic resource is useful for minimizing the peak-system temperature in a system with simple active/idle power modes. So, we addressed the objective function of minimizing the maximum resource usages for a set of application modes.

10.1.6 Radar Simulation

We model an automotive adaptive cruise control system using the frequency modulated continuous wave (FMCW) technique [46]. This experiment performs range and Doppler estimation of a moving vehicle. The radar system opportunistically (as specified in the mode) estimates the distance between the vehicle it is mounted on and the vehicle in front of it, and alerts the driver when the two become too close.

10.2 Future Work

Mixed criticality and multiprocessors are the two promising features which received considerable research attention in real-time community over the past few years. Multi-modal systems powered with either multiprocessor or mixed-criticality tasks will be very effective tool for system designer to model dynamic systems with timing constraints.

10.2.1 MMS with mixed-criticality tasks systems

In order to reduce the size and cost of an embedded system, real-time researchers have been considering multiple components (e.g., cruise control, anti-lock braking system, and audio/video for automotive systems) to be assembled upon a single computing device. These components may not share the same level of criticality (e.g., the criticality of ABS is higher than onboard audio system). For such systems, different levels of certification of the system with varying degrees of rigorousness are desired. Mixed-criticality of a task is represented by different worst case execution time (WCET) at different critical levels. Whenever a higher priority task has its execution time that exceeds its WCET at the current criticality level, the system changes the criticality level to the next higher level, and discards all jobs with criticality less than the current criticality level. With these changes in criticality level, essentially changes in computation requirements occur which may be modeled by real-time multi-modal systems. In this project, we will consider the state-of-the-art for multi-modal systems and corresponding schedulability analysis as an effective tool for analyzing mixed criticality tasks systems.

10.2.2 MMS for multiprocessors

Schedulability analysis for a multi-modal system upon a multiprocessor is going to be studied. Each mode associates a software component with hardware requirements. Software component is modeled by constrained deadline sporadic tasks, whereas the hardware part will be modeled more-general multiprocessor periodic resource model [69] or bounded-delay multi-partition [48]. For better understanding, we start with a simple hardware model (a set of bounded-delay uniprocessors where delay equals to zero) for the multiprocessor. This assumption also reduces pessimism. Each mode executes the workload upon the underlying hardware using EDF. Our goal is to design schedulable multi-modal systems upon multiprocessor, which are eventually suitable for control systems. Control systems requires typically higher number of modes, therefore, a schedulability analysis with pseudo-polynomial complexity is desirable for such systems to reduce system design time.

LIST OF PUBLICATIONS

JOURNAL

1. Masud Ahmed, Nathan Fisher, Tractable Schedulability Analysis and Resource Allocation for Real-time Multimodal Systems, ACM Transactions on Embedded Computing Systems., Volume 13, Issue 2s, January 2014.
2. Pradeep M. Hettiarachchi, Nathan Fisher, Masud Ahmed, Le Yi Wang, Shinan Wang, and Weisong Shi. The Design and Analysis of Thermal-Resilient Hard-Real-Time Systems, *ACM Transactions on Embedded Computing Systems*, 2013, To Appear.
3. Masud Ahmed, Nathan Fisher, Shengquan Wang, and Pradeep Hettiarachchi. Minimizing peak temperature in embedded real-time systems via thermal-aware periodic resources, *Sustainable Computing: Informatics and Systems*, 3(1), pp. 226-240, 2011.

CONFERENCE & WORKSHOP

1. Masud Ahmed, Pradeep M. Hettiarachchi, and Nathan Fisher. Analysis for Real-Time Multi-Modal FP-Scheduled Systems with Non-Preemptible Regions, Submitted in *RTAS*, 2014.
2. Nathan Fisher, Masud Ahmed, and Pradeep M. Hettiarachchi. Open Problems in Multi-Modal Scheduling Theory for Thermal-Resilient Multicore Systems, 5th Real-Time Scheduling Open Problems Seminar (RTSOPS), Madrid, Spain, July, 2014. (The "most wanted problem" award)
3. Masud Ahmed, Safraz Rampersaud, Nathan Fisher, Daniel Grosu and Loren Schwiebert., GPU-Based Parallel EDF-Schedulability Analysis of Multi-Modal Real-Time Systems, Proc. of 15th IEEE International Conference on High Performance Computing and Communications, November, 2013, Zhangjiajie, China.

4. Masud Ahmed, Camille Williams, Bo Peng, and Nathan Fisher. Real-Time Multi-Modal Implementation of a Robotic Toy Car. The 2nd Open Demo Session of Real-Time Systems (RTSS@Work), Puerto-Rico, 2012.

5. Masud Ahmed and Nathan Fisher and Daniel Grosu. A Parallel Algorithm for EDF-Schedulability Analysis of Multi-Modal Real Time Systems, *Proc. of the 18th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Seoul, 2012.

6. Pradeep M. Hettiarachchi and Nathan Fisher and Masud Ahmed and Le Yi Wang and Shinan Wang and Weisong Shi. The Design and Analysis of Thermal-Resilient Hard-Real-Time Systems, *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symp.*, Beijing, 2012.

7. Nathan Fisher and Masud Ahmed. Tractable Real-Time Schedulability Analysis for Mode Changes under Temporal Isolation, *Proc. of the 9th IEEE Symp. on Embedded Systems for Real-Time Multimedia*, Taipei, October 2011. (**Best paper candidate award**)

REFERENCES

[1] Luca Abeni and Giorgio Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the Real-Time Systems Symposium*, pages 3–13, Spain, December 1998. IEEE Computer Society Press.

[2] Luca Abeni and Giorgio Buttazzo. Resource reservation in dynamic real-time systems. *Real-Time Systems*, 27(2):70–77, 1999.

[3] S.G. Abraham and S.A. Mahlke. Automatic and efficient evaluation of memory hierarchies for embedded systems. In *Proc. of 32nd Annual International Sympages on Microarchitecture*, pages 114 –125, 1999.

[4] Masud Ahmed and Nathan Fisher. Tractable schedulability analysis and resource allocation for real-time multi-modal systems. *ACM Trans. Embed. Comput. Syst.*, 13(2s), 2014.

[5] Masud Ahmed, Nathan Fisher, and Daniel Grosu. A parallel algorithm for edf-schedulability analysis of multi-modal real time systems. Technical report, 2012.

[6] Masud Ahmed, Nathan Fisher, Shengquan Wang, and Pradeep Hettiarachchi. Minimizing peak temperature in embedded real-time systems via thermal-aware periodic resources. *Sustainable Computing: Informatics and Systems*, 3(1):226–240, 2011.

[7] Samar Al-Khairi. Power management features on the embedded ultra low-power intel486tm processor. Technical report, Intel, IL, USA, 1997. Available at http://www.intel.com/design/intarch/papers/lp486.htm.

[8] S. Baruah, R. Howell, and L. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems: The International Journal of Time-Critical Computing*, 2:301–324, 1990.

[9] S. Baruah, R. Howell, and L. Rosier. Feasibility problems for recurring tasks on one processor. *Theoretical Computer Science*, 118(1):3–20, 1993.

[10] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium*, pages 182–190, Orlando, Florida, 1990. IEEE Computer Society Press.

[11] Sanjoy Baruah. The limited-preemption uniprocessor scheduling of sporadic task systems. In *Real-Time Systems, 2005.(ECRTS 2005). Proceedings. 17th Euromicro Conference on*, pages 137–144. IEEE, 2005.

[12] Sanjoy Baruah. Schedulability analysis of global deadline-monotonic scheduling. Unpublished manuscript, Available at `http://www.cs.unc.edu/~baruah`, 2007.

[13] Sanjoy Baruah and Shun-Shii Lin. Pfair scheduling of generalized pinwheel task systems. *IEEE Transactions on Computers*, 47(7), July 1998.

[14] M. Behnam, T. Nolte, and R.J. Bril. Tighter schedulability analysis of synchronization protocols based on overrun without payback for hierarchical scheduling frameworks. In *Engineering of Complex Computer Systems (ICECCS), 16th IEEE International Conference on*, pages 35–44, April 2011.

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

[16] E. Bini and G. Buttazzo. Biasing effects in schedulability measures. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, pages 196–203. IEEE Computer Society, 2004.

[17] Enrico Bini and Giorgio Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers*, 53(11):1462–1473, 2004.

[18] S. Brandt, S. Banachowski, C. Lin, and T. Bisson. Dynamic integrated scheduling of hard real-time , soft real-time and non-real-time processes. In *Proceedings of the 24th IEEE Real-Time Systems Symposium*, pages 396–407, 2003.

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

[20] Alan Burns, Andy J Wellings, CM Bailey, and E Fyfe. The olympus attitude and orbital control system a case study in hard real-time system design and implementation. In *Ada-Europe'93*, pages 19–35. Springer, 1993.

[21] Giorigo Buttazzo and Luca Abeni. Adaptive rate control through elastic scheduling. In *Proceedings of the 24th IEEE Conference on Decision and Control*, volume 5, pages 4883–4888, 2000.

[22] T. L. Casavant and J. G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Trans. Softw. Eng.*, 14(2):141–154, 1988.

[23] Thidapat Chantem, Robert P. Dick, and X. Sharon Hu. Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs. In *Design, Automation and Test in Europe*, pages 288–293, 2008.

[24] Thidapat Chantemand, Hu X. Sharon, and Robert P. Dick. Online work maximization under a peak temperature constraint. In *ISLPED '09: Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design*, pages 105–110, New York, NY, USA, 2009. ACM.

[25] Jian-Jia Chen, Chia-Mei Hung, and Tei-Wei Kuo. On the minimization of the instantaneous temperature for periodic real-time tasks. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 236–248, 2007.

[26] Jian-Jia Chen, Shengquan Wang, and Lothar Thiele. Proactive speed scheduling for frame-based real-time tasks under thermal constraints. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2009.

[27] Michele Cirinei and Theodore P. Baker. Edzl scheduling analysis. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, Pisa, Italy, July 2007. IEEE Computer Society Press.

[28] Control theory. Control theory — Wikipedia, the free encyclopedia, 2010. [Online; accessed September 5, 2014].

[29] Robert L. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. Controller area network (CAN) schedulability analysis: Refuted, revisited, and revised. volume 35, pages 239–272. Kluwer Academic Publishers, 2007.

[30] Michael L. Dertouzos. Control robotics: The procedural control of physical processes. In *IFIP Congress*, pages 807–813, 1974.

[31] Derek L. Eager, Edward D. Lazowska, and John Zahorjan. Adaptive load sharing in homogeneous distributed systems. *IEEE Trans. Softw. Eng.*, 12(5):662–675, 1986.

[32] Arvind Easwaran, Madhukar Anand, and Insup Lee. Compositional analysis framework using EDP resource models. In *Proceedings of the IEEE Real-time Systems Symposium*, Tuscon, Arizona, December 2007.

[33] Jimin Feng, S. Chakraborty, B. Schmidt, Unmesh Weiguo Liu, and U.D. Bordoloi. Fast schedulability analysis using commodity graphics hardware. In *Proc. of 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 400–408, 2007.

[34] Xiang (Alex) Feng and Al Mok. A model of hierarchical real-time virtual resources. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 26–35. IEEE Computer Society, 2002.

[35] Nathan Fisher and Masud Ahmed. Tractable real-time schedulability analysis for mode changes under temporal isolation. In *Proc. of the 9th IEEE Sympages on Embedded Systems for Real-Time Multimedia*, Taipei, 2011.

[36] Xing Fu, Xiaorui Wang, and Eric Puster. Simultaneous thermal and timeliness guarantees in distributed real-time embedded systems. *Journal of Systems Architecture*, 2010. To Appear.

[37] Yong Fu, Nicholas Kottenstette, Yingming Chen, Chenyang Lu, Xenofon D. Koutsoukos, and Hongan Wang. Feedback thermal control for real-time system. In *Proceedings of the Real-Time and Embedded Technology and Applications Systems Symposium*, Stockholm, Sweden, April 2010. IEEE Computer Society Press.

[38] Yong Fu, Nicholas Kottenstette, Chenyang Lu, and Xenofon D. Koutsoukos. Feedback thermal control of real-time systems on multicore processors. In *Proceedings of the Tenth ACM International Conference on Embedded Software*, pages 113–122, New York, NY, USA, 2012. ACM.

[39] Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. *Introduction to Parallel Computing, Second Edition*. Addison Wesley, USA, 2003.

[40] Qian Guangming. An earlier time for inserting and/or accelerating tasks. *Real-Time Systems*, 41:181–194, 2009.

[41] P.M. Hettiarachchi, N. Fisher, and L.Y. Wang. Achieving thermal-resiliency for multicore hard-real-time systems. In *Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on*, pages 37–46, July 2013.

[42] Pradeep M. Hettiarachchi, Nathan Fisher, Masud Ahmed, Le Yi Wang, Shinan Wang, and Weisong Shi. The design and analysis of thermal-resilient hard-real-time systems. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Sympages*, Beijing, 2012.

[43] Wei-Lun Hung, Yuan Xie, Narayanan Vijaykrishnan, Mahmut T. Kandemir, and Mary Jane Irwin. Thermal-aware task allocation and scheduling for embedded systems. In *ACM/IEEE Conference of Design, Automation, and Test in Europe*, pages 898–899. IEEE Computer Society, 2005.

[44] R. Inam, M. Sjodin, and R.J. Bril. Mode-change mechanisms support for hierarchical freertos implementation. In *Emerging Technologies Factory Automation (ETFA), 2013 IEEE 18th Conference on*, pages 1–10, 2013.

[45] D. S. Johnson. *Near-optimal Bin Packing Algorithms*. PhD thesis, Department of Mathematics, Massachusetts Institute of Technology, 1973.

[46] C. Karnfelt. 77 ghz acc radar simulation platform. In *IEEE International Conferences on Intelligent Transport Systems Telecommunications (ITST)*, 2009.

[47] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the Real-Time Systems Symposium - 1989*, pages 166–171, Santa Monica, California, USA, December 1989. IEEE Computer Society Press.

[48] Giuseppe Lipari and Enrico Bini. A framework for hierarchical scheduling on multiprocessors: from application requirements to run-time allocation. In *Proc. of the IEEE Real-Time Systems Symp.* IEEE Computer Society, 2010.

[49] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[50] David Meisner, Brian T. Gold, and Thomas F. Wenisch. Powernap: eliminating server idle power. In *Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*, ASPLOS ’09, pages 205–216, New York, NY, USA, 2009. ACM.

[51] A. K. Mok. *Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment*. PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983. Available as Technical Report No. MIT/LCS/TR-297.

[52] Srinivasan Murali, Almir Mutapcic, David Atienza, Rajesh Gupta, Stephen Boyd, Luca Benini, and Giovanni De Micheli. Temperature control of high-performance multi-core platforms using convex optimization. In *DATE*, pages 110–115, 2008.

[53] Srinivasan Murali, Almir Mutapcic, David Atienza, Rajesh Gupta, Stephen Boyd, and Giovanni De Micheli. Temperature-aware processor frequency assignment for mpsocs using convex optimization. In *IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 111–116, New York, NY, USA, 2007. ACM.

[54] Vincent Nelis and Joel Goossens. Mode change protocol for multi-mode real-time systems upon identical multiprocessors. *Proceeding of the Euromicro Conference on Real-Time Systems*, 0:151–160, 2009.

[55] Vincent Nelis, Joel Goossens, and Bjorn Andersson. Two protocols for scheduling multi-mode real-time systems upon identical multiprocessor platforms. *Real-Time Systems, Euromicro Conference on*, 0, 2008.

[56] S. Nunna, U.D. Bordoloi, S. Chakraborty, pages Eles, and Zebo Peng. Exploiting gpu on-chip shared memory for accelerating schedulability analysis. In *Proc. of the International Sympages on Electronic System Design (ISED)*, pages 147 –152, 2010.

[57] NVIDIA. Nvidias next generation CUDA compute architecture: Fermi, Feb 2010. Available at http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute _Architecture_Whitepaper.pdf.

[58] NVIDIA. NVIDIA CUDA C programming guide, Feb 2011. Available at http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Pro gramming_Guide.pdf.

[59] NVIDIA. CUDA in action - research and apps, Feb 2012. Available at http://developer.nvidia.com/cuda–action–research–apps.

[60] P. Pedro and A. Burns. Schedulability analysis for mode changes in flexible real-time systems. In *Proc. of the 10th Euromicro Workshop on Real-Time Systems*, pages 172–179, Berlin, 1998.

[61] Linh T. X. Phan and Insup Lee. Towards a compositional multi-modal framework for adaptive cyber-physical systems. In *Proc. of the 17th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 67–73. IEEE Computer Society, 2011.

[62] L.T.X. Phan, S. Chakraborty, and I. Lee. Timing analysis of mixed time/event-triggered multi-mode systems. In *Proc. of the IEEE Real-Time Systems Sympages*, pages 271–280, December 2009.

[63] L.T.X. Phan, Insup Lee, and O. Sokolsky. Compositional analysis of multi-mode systems. In *Proc. of the 22nd Euromicro Conference on Real-Time Systems*, pages 197–206, Brussels, 2010.

[64] Jorge Real and Alfons Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, 26(2):161–197, 2004.

[65] MATLAB Phased Array Toolbox. AACC using FMCW Technology, www.mathworks.com/help/phased/examples/automotive-adaptive-cruise-control-using-fmcw-technology.html, 2014.

[66] Luca Santinelli, Giorgio Buttazzo, and Enrico Bini. Multi-moded resource reservations. In *Proc. of the 17th IEEE Real-Time and Embedded Technology and Applications Sympages*, pages 37–46, Chicago, 2011.

[67] Lui Sha, Ragunathan Rajkumar, Ragunathan Rajkumar, John Lehoczky, John Lehoczky, Krithi Ramamritham, and Krithi Ramamritham. Mode change protocols for priority-driven preemptive scheduling. *Real-Time Systems*, 1:243–264, 1988.

[68] Eugene Shih, Paramvir Bahl, and Michael J. Sinclair. Wake on wireless: an event driven energy saving strategy for battery operated devices. In *Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 160–171, Atlanta, Georgia, USA, 2002. ACM, New York, NY, USA.

[69] Insik Shin, Arvind Easwaran, and Insup Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, Prague, Czech Republic, July 2008. IEEE Computer Society Press.

[70] Insik Shin and Insup Lee. Compositional real-time scheduling framework with periodic model. *ACM Transactions on Embedded Computing Systems*, 7(3), April 2008.

[71] D.G. Smith, R.C. Dixon, and J.S. Vanderpool. Multi-band, multi-mode spread-spectrum communication system, December 2 1997. US Patent 5,694,414.

[72] B. Sprunt, L. Sha, and J. P. Lehoczky. Aperiodic task scheduling for hard real-time systems. *Real-Time Systems*, 1:27–69, 1989.

[73] John A. Stankovic, Marco Spuri, Krithi Ramamritham, and Giorgio C. Buttazzo. *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*. Kluwer Academic Publishers, 101 Philip Drive, Assinippi Park Norwell, MA 02061, USA, 1998.

[74] Nikolay Stoimenov, Simon Perathoner, and Lothar Thiele. Reliable mode changes in real-time systems with fixed priority or edf scheduling. In *Proc. of the Conference on Design, Automation and Test in Europe*, pages 99–104, France, 2009.

[75] Nikolay Stoimenov, Lothar Thiele, Luca Santinelli, and Giorgio Buttazzo. Resource adaptations with servers for hard real-time systems. In *Proc. of the 10th ACM International Conf. on Embedded Software*, pages 269–278, New York, 2010.

[76] K. W. Tindell and A. Alonso. A very simple protocol for mode changes in priority preemptive systems. Technical report, Universidad Politecnica de Madrid, 1996.

[77] K.W. Tindell, A. Burns, and A.J. Wellings. Mode changes in priority preemptively scheduled systems. In *Proc. of the 13th IEEE Real-Time Systems Sympages*, pages 100 –109, Arizona, 1992.

[78] Manish Vachharajani. *Microarchitecture Modeling for Design-Space Exploration*. PhD thesis, Princeton University,USA, 2004.

[79] S. Wang and R. Bettati. Delay analysis in temperature-constrained hard real-time systems with general task arrivals. In *IEEE Real-Time Systems Symposium*, 2006.

[80] S. Wang and R. Bettati. Reactive speed control in temperature-constrained real-time systems. In *Euromicro Conference on Real-Time Systems*, 2006.

[81] S. Wang and R. Bettati. Reactive speed control in temperature-constrained real-time systems. *Real-Time Systems Journal*, 39(1-3):658–671, 2008.

[82] Shucai Xiao and Wu-Chun Feng. Inter-block gpu communication via fast barrier synchronization. In *Proc. of the IEEE Parallel and Distributed Processing Symposium*, pages 1–12, Atlanta, 2010.

ABSTRACT

THE DESIGN, ANALYSIS, & APPLICATION OF MULTI-MODAL REAL-TIME EMBEDDED SYSTEMS

by

MASUD AHMED

December 2014

Advisor: Dr. Nathan Fisher

Major: Computer Science

Degree: Doctor of Philosophy

For many hand-held computing devices (e.g., smartphones, tablet computers, and GPS receivers), multiple operational modes are preferred because of their flexibility. In addition to their designated purposes, some of these devices provide a platform for different types of services, which include rendering of high-quality multimedia. Upon such devices, temporal isolation among co-executing applications is very important to ensure that each application receives an acceptable level of quality-of-service. In order to provide strong guarantees on services, multimedia applications and real-time control systems maintain timing constraints in the form of deadlines for recurring tasks. A flexible real-time multi-modal system will ideally provide system designers the option to change both resource-level modes and application-level modes. Existing schedulability analysis for a real-time multi-modal system (MMS) with software/hardware modes are computationally intractable. In addition, a fast schedulability analysis is desirable in a design-space exploration that determines the "best" parameters of a multi-modal system by repeated application of the MMS schedulability analysis. The thesis of this dissertation is:

The determination of resource parameters with guaranteed schedulability for real-time systems that may change computational requirements over time is expensive in terms of runtime. However, decoupling schedulability analysis from determining the

minimum processing resource parameters of a real-time multi-modal system results in pseudo-polynomial complexity for the combined goals of determining both MMS schedulability and optimal resource parameters.

Effective schedulability analysis and optimized resource usages are essential for an MMS that may co-execute with other application upon a low-end shared platform to reduce size and cost of an embedded system. Traditional real-time systems research has commonly addressed the issues of schedulability under mode changes and temporal isolation, separately and independently. For instance, schedulability analysis of real-time multi-mode systems has commonly assumed that the system is executing upon a dedicated processing platform. On the other hand, research on temporal isolation in real-time scheduling (often called *server-based* or *hierarchical* scheduling), while permitting the analysis of real-time subsystems that co-execute upon a shared computation platform, has often assumed that the application and resource requirements of each subsystem are fixed during runtime. Only recently have researchers started to address the problem of guaranteeing hard deadlines of temporally-isolated subsystems in multi-modal systems. However, most of this recent research suffers from two fundamental drawbacks: 1) full support for both resource-level mode changes or application-level mode changes does not exist, and/or 2) the proposed algorithms for determining schedulability under mode changes have exponential-time complexity. As a result, current literature on multi-modal systems cannot guarantee optimal resource usages under mode changes. In this dissertation, first we address the two fundamental drawbacks by providing a theoretical framework and associated tractable schedulability analysis for hard real-time subsystems executing upon a temporally-isolated environment under both resource and application-level mode changes. Then, by leveraging the developed schedulability analysis, we address the problem of optimizing a multi-mode real-time system with respect to resource usages over all modes.

To accelerate the schedulability analysis of a multi-modal system, as well as determination of the minimum parameters, we develop a parallel algorithm using message passing parallel systems to check the invariants of the schedulable real-time MMS. This parallel algorithm significantly improves the execution time for checking the schedulability of a single set of parameters (e.g., our parallel algorithm requires only approximately 45 minutes to analyze a 16-mode system upon 8 cores, whereas the analysis takes 9 hours when executed on a single core). However, even this reduction is still expensive for applying techniques such as design-space exploration (DSE) that repeatedly applies schedulability analysis to determine the optimal system resource parameters. Today's massively parallel GPU platforms can be a cost-effective alternative to scaling the number of computer nodes and further reducing the computation time for multi-modal schedulability analysis. As massively-parallel Graphical Processing Units (GPU) are increasingly common for handheld devices, an efficient GPU-based schedulability analysis can also be used online to reconfigure the system by re-evaluating schedulability if parameters change dynamically. In this dissertation, we also extend our parallel schedulability analysis algorithm to a GPU implementation. Finally, we performed a case-study of radar-assisted cruise control system to show the usability of multimode system which consists of fixed priority non-preemptive tasks.

AUTOBIOGRAPHICAL STATEMENT

Masud Ahmed completed his B.Sc.Engg degree in Computer Science & Engineering (Electrical Engineering as minor) on June 2005 from Bangladesh University of Engineering & Technology (BUET) which is the top university in Bangladesh. He joined Commlink Info Tech Ltd, Dhaka as Member, R&D, and promoted to Senior Member, R&D. He left Commlink for graduate studies starting from Fall 2008 at Wayne State University.

Masud Ahmed received his M.S. degree in Computer Science from Wayne State University, Detroit, Michigan on 2011. He completed his Ph.D. in Computer Science at Wayne State University under the supervision of Prof. Nathan W. Fisher on December 2014. His research interests include real-time systems, embedded systems, and parallel computing. While pursuing graduate degrees, he worked as Graduate Teaching Assistant and Graduate Research Assistant in Computer Science department.

Masud started working for Mathworks from January 2013. He worked as a Software Engineer Intern at Carsley & Associates in Summer 2010. He was a student member of ACM and IEEE.