Latency, energy, and schedulability of real-time embedded systems

Liu, D.; Liu D.

Latency, Energy, and Schedulability of Real-Time Embedded Systems

Di Liu

Latency, Energy, and Schedulability of Real-Time Embedded Systems

PROEFSCHRIFT

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van Rector Magnificus Prof.mr. C.J.J.M. Stolker,
volgens besluit van het College voor Promoties
te verdedigen op woensdag 6 september 2017 klokke 13:45 uur

door

Di Liu
geboren te Lijiang, China
in 1984

Promotor:	Prof. Dr. Joost N. Kok	Universiteit Leiden
Co-Promotor:	Dr. Todor P. Stefanov	Universiteit Leiden
Promotion Committee:	Prof. Dr. Koen Langendoen	Technische Universiteit Delft
	Prof. Dr. Kai Huang	Sun Yat-Sen University, China
	Dr. Sebastian Altmeyer	Universiteit van Amsterdam
	Prof. Dr. Aske Plaat	Universiteit Leiden
	Prof. Dr. Thomas Bäck	Universiteit Leiden
	Prof. Dr. Harry A.G. Wijshoff	Universiteit Leiden

Latency, Energy, and Schedulability of Real-Time Embedded Systems
Di Liu. -
Dissertation Universiteit Leiden. - With ref. - With summary in Dutch.

This dissertation was typeset using LaTeX in Linux and version controlled using Git.

ISBN: 978-94-6299-658-8
Printed by: Ridderprint, the Netherlands.

Contents

List of Tables

List of Figures

List of Abberivations

Proposed Algorithms

ASHM Allocation and Split on Heterogeneous Multicore

DM Density Minimization

FDM Frequency Driven Mapping

Data-Flow Models

CSDF Cyclo-Static Data-Flow

SDF Synchronous Data-Flow

Mixed-Criticality Systems

AMC Adaptive Mixed-Criticality

CA Certification Authority

EDF-VD Earliest Deadline First with Virtual Deadline

IMC Imprecise Mixed-Criticality

MC Mixed-Criticality

Others

ASIP Application Specific Instruction Processor

EE Energy-Efficient

ICP Integer Convex Programing

ILP Integer Linear Programing

MPSoC Multi-Processor System-on-Chip

PE Performance-Efficient

UAV Unmanned Aerial Vehicle

VFS Voltage and Frequency Scaling

Real-Time Systems

BF Best-Fit

DBF Demand Bound Function

EDF Earliest Deadline First

FF First-Fit

FFD First-Fit-Decreasing

HRT Hard-Real-Time

LLF Least Laxity First

QPA Quick convergency Processor-demand Analysis

WCET Worst-Case Execution Time

WF Worst-Fit

WFD Worst-Fit-Decreasing

Chapter 1

Introduction

> Almost all computer systems of the future will utilize real-time scientific principles and technology.
>
> John Stankovic

EMBEDDED systems are computer systems dedicated to a specific functionality. Usually embedded systems are integrated into a large and complex system to control, monitor and assist the operation of the whole system, safely and reliably. In some cases, we may not be aware of the presence of embedded systems, but in our daily life they are prevalent and almost everywhere, affecting and somehow changing our life. From the ARTEMIS report [ART14], 98% of all computing chips are for embedded systems. A smart watch which monitors our health situation is an embedded system; A mouse which we use to control a computer is an embedded system; A thermostat which senses the temperature and humidity of our apartment is also an embedded system. We can give uncountable examples of such embedded systems which are playing an important role in our daily life without notice.

Since embedded systems usually execute control applications which constantly interact with the physical world via sensors and actuators, embedded systems are required to execute not only functionally correctly but also on time. This critical requirement related to time is referred as *real-time constraints* [But11]. Systems are called real-time systems, if the correctness of the system does not only depend on the correctness of the system output but also on whether the output is delivered on time [Sta88]. Based on consequences of missing time deadlines by an application, real-time systems are categorized into two types:

- **Hard-real-time** systems: missing deadlines will lead to failure of the system which in turn causes catastrophic consequences, e.g., loss of human life. Exam-

ples of *hard-real-time* systems are safety control systems in cars and aircrafts, pacemakers, etc;

- **Soft-real-time** systems: missing deadlines will not cause failure of the whole system but will degrade the performance of the system. Examples of *soft-real-time* systems are multimedia applications, on-line service applications, etc.

Embedded systems with real-time constraints are called *real-time embedded systems*.

Besides real-time constraints, many applications of real-time embedded systems feature another important property, called *criticality*. The *criticality* is to denote degrees of importance in guaranteeing the safety of a system. For example, unmanned aerial vehicles (UAVs) have two types of applications, safety-critical applications, such as the flight control, and mission-critical applications, such as surveillance and video streaming. The safety-critical applications (e.g., the flight control) have higher criticality level because they are essentially crucial to the operational safety of the whole system and failure (i.e., violating timing properties) of the safety-critical applications will lead to a catastrophic consequence, such as loss of UAV which may injure a human-being. On the other hand, the mission-critical applications have lower criticality level because they are not coupled to the operational safety of the whole system, so failure of mission-critical applications will not threaten the operational safety of the system but will only affect the system service quality. In different industrial contexts, different standards are deployed to guide the design of systems with different criticality-level applications, such as IEC61508 for electrical/electronic/programmable electronic safety-related systems, ISO26262 for automotive systems, and DO-178B/C for avionic systems [ENNT15]. Table 1.1 defines the classification of criticality levels in standard DO-178B/C [Nor], where 5 criticality levels are present and the criticality levels are classified with respect to the failure consequence on the system safety. Criticality level A is the most critical level and failure of an A-level application leads to catastrophic consequences, whereas failure of an E-level application does not have an effect on the system safety.

Up to this point, we have introduced embedded systems, real-time constraints tightly coupled to embedded systems, and the criticality concept of real-time embedded systems. In the next section, we will discuss three significant development trends in designing real-time embedded systems.

1.1 Development Trends in Real-Time Embedded Systems

In the past decade, we have been witnessing some important development trends in the computing world which have profound effects on the design of real-time embedded systems.

Level	Failure Condition	Failure Consequence
A	Catastrophic	Failure may cause multiple fatalities, usually with loss of the airplane.
B	Hazardous	Failure has a large negative impact on safety or performance, or reduces the ability of the crew to operate the aircraft due to physical distress or a higher workload, or causes serious or fatal injuries among the passengers.
C	Major	Failure significantly reduces the safety margin or significantly increases crew workload. May result in passenger discomfort (or even minor injuries).
D	Minor	Failure slightly reduces the safety margin or slightly increases crew workload. Examples might include causing passenger inconvenience or a routine flight plan change.
E	No Effect	Failure has no impact on safety, aircraft operation, or crew workload.

Table 1.1: Criticality levels in the DO-178B/C standard [Nor]

1.1.1 The Era of Multicore/Multiprocessor Systems

When single processor systems were dominating the chip market a decade ago, chip manufacturers used to improve the performance of a processor by scaling up the operational clock frequency. Meanwhile, the fast development of the process technology enables semiconductor manufacturers to produce thiner *transistors*, the fundamental element to implement electronic circuits. Nowadays, some high-end processors, like Samsung Exynos 7 Octa 7420, 7870 and 8890, are implemented with 14 nanometer transistors [Sam16]. However, constantly scaling up the operational clock frequency of thin transistors results in extremely high power consumption [EET04].

As a solution to such high power consumption, chip manufacturers have drastically changed their design scheme from a single processor chip with high operational clock frequency to a chip with multiple cores/processors, but each with lower operational clock frequency. Fabricating more cores on a chip is able to enhance the peak performance of the system and at the same time to reduce the total power consumption in comparison to the single processor design. Throughout this dissertation, we may use the term multicore and multiprocessor interchangeably.

The year 2004 marked the milestone of this significant change in industry, when Intel canceled its single processor design, namely *Tejas*, and moved to a duel-core

design [EET04]. Since then, computing systems including embedded systems have entered the multicore era. Nowadays, multicore systems are the mainstream in computing systems. This trend can be seen on diverse computing systems such as mobile phones, laptops, desktops, etc.

1.1.2 The Shift to Heterogeneous Multicore Systems

Multicore systems have been widely adopted to satisfy the increasing computational demands of complicated applications and, in the meantime, to reduce energy consumption. Among all multicore systems, homogeneous multicore systems that consist of identical processing cores are most ubiquitous and widely-used in modern electronic systems spanning from mobile devices to supercomputing systems. However, the rapid development of multicore systems brings a new problem, called the *dark silicon problem* [EBSA+11]. In 1974, Dennard *et al.* [DGR+74] stated that as transistors decrease size, the power density still remains a constant, i.e., the transistors become thinner, and the power consumption also scales down along with the reduced size. This statement is wildly known as the "*Dennard Scaling*". However, when the transistor manufacturing technology enters the era of nanometer, the *Dennard Scaling* fails due to the dramatically increased static power consumption in nanometer-size transistors. Static power is consumed by currents which leak through transistors even when transistors are turned off [KAB+03]. This significant increase in static power consumption in turns leads to an overheating issue for the system. To avoid the overheating, some transistors on a chip have to be inactive (powered-off), i.e., *'dark'*.

Several solutions [CZZ+15, HKPS15] have been proposed in recent years to mitigate the dark silicon problem. Heterogeneous multicore systems [Mit15] have been considered to be one of the promising solutions for the dark silicon problem and a good alternative to homogeneous multicore systems. In contrast to homogeneous systems having identical cores, heterogeneous multicore systems consist of different types of cores. Such variety of cores enable diverse applications to enhance the application performance and/or reduce the power/energy consumption of the application by means of selecting a proper core for execution.

Among all heterogeneous systems, the single-ISA heterogeneous multicore system [KFJ+03] is a special type of heterogeneous multicore system, where the cores on the chip have the same *instruction set architecture* (ISA) but differentiate with each other in terms of power consumption and performance. Typical single-ISA heterogeneous multicore systems usually consist of two types of cores; 'big' cores with complex micro-architecture, e.g., a deep pipeline and wider issue width, designed for high performance computing and 'LITTLE' cores with simple micro-architecture, e.g., a shallow pipeline and narrower issue width, optimized for low power computing. Table 1.2 shows an example of cores implemented on a 'big.LITTLE' architecture

Core type	pipeline depth	Out-of-order execution	Decode	big.LITTLE role
ARM Cortex A57	15	Yes	3-wide-issue	'big'
ARM Cortex A53	8	No	2-wide-issue	'LITTLE'

Table 1.2: Comparison between ARM Cortex A57 and ARM Cortex A53 [ARM16]

system and gives a comparison of the two types of ARM cores in terms of microarchitecture [ARM16]. Several leading semiconductor companies have mass-produced their own single-ISA heterogeneous multicore systems for commodity products, e.g., Qualcomm Snapdargon 810 and 808, Samsung's Exynos 5 Octa series [Sam16], and Nvidia's Tegra X1[Gil15]. In the remainder of this dissertation, when we refer to heterogeneous multicore/multiprocessor systems, we mean single-ISA heterogeneous multicore/multiprocessor systems.

1.1.3 The Emergence of Mixed-Criticality Systems

Real-time systems which execute applications with different criticality are becoming prevalent, e.g, automotive vehicles, unmanned aerial vehicles, aircrafts, etc. To ensure the safety guarantee of systems with different critical-level applications, the old paradigm of designing such safety-critical systems was to physically isolate applications with different criticality level, i.e., critical applications and non-critical applications are executed separately on different processing units. Such complete spatial isolation enables critical applications to avoid the interference from non-critical applications, thereby guaranteeing system safety. However, with the rapid development of complex and sophisticated real-time systems, increasing number of applications with different criticality and complex functionality are incorporated into a system, leading to a huge growing number of processing units. For instance, modern premium cars typical contain around 70-100 computers, around 100 electronic motors and 2 km of wire [Tho12]. This complicated and sometimes redundant hardware leads to a system with large system size and very high power consumption. Therefore, to reduce Size, Weight, and Power (SWaP), the emerging trend in the development of safety-critical systems is to integrate applications with different criticality into a shared computing platform. We call such systems *mixed-criticality systems*. A formal definition of a *mixed-criticality system* is given as follows:

Definition 1.1.1 ([BBB+09])**.** A mixed-criticality system is an integrated suite of hardware, operating system and middleware services, and application software that supports the execution of safety-critical, mission-critical, and non-critical software within a single, secure compute platform.

1.2 Problem Statement

The important development trends, described in Section 1.1, bring new opportunities to develop embedded systems, but they also arise several challenges when designing real-time embedded systems. In this dissertation, we address challenges arisen by the above-mentioned development trends in the contexts of system resource optimization, system energy optimization, and system schedulability analysis. The specific problems, we address in this dissertation, are formulated as follows.

Problem 1: Resource Optimization for hard-real-time streaming applications.

Streaming applications, such as video/audio processing and digital signal processing, have become prevalent in embedded systems. These applications contain ample amount of parallelism which perfectly matches the processing power of Multi-Processor System-on-Chip (MPSoC) platforms. To efficiently program MPSoC platforms, Models-of-Computation (MoCs) are usually used to specify streaming applications. Prominent examples of MoCs include Synchronous Data Flow (SDF) [LM87] and its generalization Cyclo-Static Dataflow (CSDF) [BELP96], in which actors representing computation are executed concurrently, thereby naturally exposing parallelism.

Traditionally, self-timed scheduling [MB07] is considered to be the most proper scheduling paradigm to schedule Data-Flow modeled applications. However, hard-real-time constraints are increasingly imposed to streaming applications and self-timed scheduling cannot guarantee such rigorous constraints. In addition, self-timed scheduling suffers from complex analysis techniques, making its design procedure really time-consuming. Recently, Bamakhrama and Stefanov in [BS11][BS12][BS13] proposed a scheduling framework that schedules acyclic CSDF graphs by using hard-real-time theories. In this scheduling framework, each CSDF actor executes strictly periodically and meets a given deadline. The periodic execution of actors guarantees a certain throughput and latency. Additionally the well-defined analytical techniques of real-time theories significantly reduce the design time when designing embedded multiprocessor streaming systems [BZNS12]. When CSDF actors are scheduled as strictly periodic tasks, the deadline of each actor can be varied in a well-defined bounded interval (see Section 2.3), thereby controlling the application latency and the number of processors needed to schedule the application. *This means that selecting a proper deadline value for each actor is an important issue for reducing the latency and minimizing the number of processors in this framework.* Although, in [BS12], the authors give a method to select deadlines of actors to reduce the application latency, their method is not optimal in terms of the required number of processors. **The problem, we address in the scheduling framework proposed in [BS11][BS12][BS13], is how to select deadlines of actors of hard-real-time streaming applications in**

a proper way such that the resources (the number of processors) is minimized while meeting their latency constraints.

Problem 2: Energy-efficient mapping and scheduling of hard-real-time applications on "big.LITTLE" heterogeneous multicore systems.

Energy/power consumption has gradually become a critical design criterion for a system, especially for embedded systems which are mostly battery-powered. Voltage/frequency scaling (VFS) is the most common technique for power reduction and can be seen on many modern processors. Due to its prevalence, VFS is also applied to real-time embedded systems for energy minimization.

Basically, for energy-efficient real-time application mapping, an algorithm with consideration of energy efficiency is deployed first to map real-time applications on multicore systems and then the VFS technique is used on the system to scale down the operational clock frequency/voltage to a proper level that is able to guarantee the time deadlines of all real-time applications and to minimize the energy consumption at the same time. However, the energy-efficient mapping and scheduling problem has been proven to be $\mathbf{NP}$-hard in the strong sense on homogeneous multiprocessor systems [AY03] as well as on heterogeneous multiprocessor systems [CT08]. Thus, heuristic or approximate algorithms are required to deal with the problem in a reasonable time. Many approaches have been proposed to effectively and efficiently map real-time applications in an energy-efficient manner. Two surveys in [CK07, BMAB16] comprehensively review existing works concerning energy-efficient mapping and scheduling of real-time applications.

However, the existing approaches cannot effectively handle the new heterogeneous "big.LITTLE" multicore systems discussed in Section 1.1.2, because some important features of these emerging heterogeneous multicore systems were not considered. Hence, this fact motivates us to revisit the existing mapping and scheduling approaches and consider the features of the new heterogeneous multicore systems. **The problem, we address, is how to map hard-real-time applications on the emerging "big.LITTLE" heterogeneous multicore systems in an energy-efficient manner while satisfying real-time constraints and performance requirement when considering hard-real-time streaming applications.**

Problem 3: Schedulability of imprecise mixed-criticality systems.

To ensure the correctness of a mixed-criticality (MC) system, highly critical applications are subject to certification by Certification Authorities (CAs), such as the Civil Aviation Authority [Civ16] and Federal Aviation Administration [Fed16], and usually the certifications are done under extremely rigorous and pessimistic assumptions [Ves07]. As a consequence, this pessimism generally causes large worst-case execution time (WCET) over-estimation for highly critical applications and in turn it

results in underutilization of the hardware resource.

To deal with this overestimation, Vestal proposed in [Ves07] to characterize a highly critical application with different WCETs corresponding to different criticality levels. Besides the large WCET determined by the CAs, each highly critical application is specified with several smaller WCETs which are determined by system designers at lower assurance levels, i.e., considering less pessimistic situations. Since CAs only certify highly critical applications, all less critical applications are only validated by system designers who generally do this under less pessimistic situations, thereby having only lower assurance and smaller WCETs. When scheduling an MC system modeled as described above, all applications (highly critical and lowly critical) are initially scheduled using their low assurance WCETs. This can better utilize hardware resources, and in most cases all applications can be safely and successfully scheduled with their low assurance WCETs. Then, if a rare case occurs, i.e., any highly critical application cannot complete its execution within its low assurance WCET, the system discards all less critical applications and dedicates the whole system to schedule only highly critical applications with their certified (very pessimistic, large) WCETs. Throughout this dissertation, we call the MC model discussed above the *classical* MC model.

Although the *classical* MC model captures the core features of MC systems, it also receives some criticisms from system engineers because completely discarding less critical applications is too pessimistic and in some cases unacceptable [BB13]. To address these criticisms, Burns and Baruah in [BB13] proposed a more general MC model. In this general MC model, besides the normal WCET validated by system designers, a reduced WCET is given to each less critical application. Then, if a rare situation occurs, i.e., highly critical applications overrun their low assurance WCETs, instead of discarding all less critical applications, this general MC model keeps less critical applications running with their reduced WCETs. Reducing WCETs to keep less critical applications running is conceptually similar to the *imprecise computation model* [LLS$^+$91][LSL$^+$94]. In the *imprecise computation model*, the output quality of a real-time application depends on its execution time. The longer a task executes, the better quality results it produces. Then, if there is an overload in the system, tasks can trade off the quality of the produced results to ensure their timing correctness. In [RKKK14a], Ravindran et al. give several real-life applications with this imprecise feature in different domains, e.g., video encoding, robotic control, cyber-physical systems, and planetary rovers. Considering the conceptual analogy between the general MC model proposed in [BB13] and the *imprecise model*, we call this general MC model *imprecise mixed-criticality* (IMC) model in this dissertation.

For MC real-time systems or even non-MC real-time systems (all applications have the same criticality level), the most important problem is to analyze the *schedu-*

lability (i.e., feasibility) of the system, i.e., whether under a certain scheduling algorithm, a set of real-time applications can run on a platform without violating any deadline, even in the worst case. To analyze the *schedulability* of a real-time system under a certain scheduling algorithm, a *schedulability test* is needed.

Definition 1.2.1. Given a scheduling algorithm, a hardware platform and a real-time application set, a *schedulability test* decides whether the application set is schedulable by the scheduling algorithm on the hardware platform.

In [BB13], Burns and Baruah presented a test based on Adaptive Mixed-Criticality (AMC) [BBD11] to check the schedulability of the IMC model under the fixed-priority scheduling algorithm [LL73]. However, a schedulability test of the IMC model under dynamic-priority scheduling algorithm, e.g., earliest deadline first (EDF) with virtual deadline (EDF-VD) [BBD+12], has not been addressed. Here, **the problem, we address, is how to ensure and test the schedulability of an IMC system under the EDF-VD scheduling algorithm.**

1.3 Contributions of This Dissertation

Below, we summarize our novel contributions to the problems outlined in Section 1.2.

Contribution 1: Novel approach for Resource Optimization of CSDF-modeled Streaming Applications with Latency Constraints
To address Problem 1 in Section 1.2, in the context of CSDF-modeled streaming applications and the *hard-real-time scheduling framework* proposed in [BS11][BS12] [BS13], we propose a new method to optimally select the deadlines of actors in CSDF graphs so that the required resources (i.e, the number of processors) are minimized while the latency requirement is satisfied. Our novel contributions are twofold: 1) we propose a new method to interpret the precedence relation between actors so that the parameters of actors, i.e., starting times and deadlines, can be formalized in a mathematical form; 2) based on our first contribution, we formulate our resource optimization problem as an *integer convex programing* (ICP) problem. Convex programming is a mathematical optimization problem which can be optimally solved in polynomial time [BV04]. The formulated ICP problem enables us to use an off-the-shelf convex programming solver, e.g., CVX [GB14][GB08] to solve our problem and obtain the optimal solution for our resource optimization problem. Compared to the existing approach [BS12], our ICP-based approach can effectively reduce the resource requirements for the hard-real-time streaming applications while guaranteeing the latency constraints. This contribution and experimental results are presented in Chapter 3.

Contribution 2: Novel Algorithm for Energy-Efficient Mapping of Real-Time Streaming Applications on Heterogeneous Multiprocessor System-on-Chip (MPSoC)

We address Problem 2 in Section 1.2 by proposing a novel polynomial time algorithm, called Frequency Driven Mapping (FDM), to map real-time streaming applications onto a heterogeneous multiprocessor system with the aim of reducing the energy consumption and guaranteeing the latency and throughput constraints. The main novelty in this algorithm is twofold: 1) By using the *hard-real-time scheduling framework* for CSDF graphs in [BS11][BS12][BS13], we propose an efficient way to determine a suitable processor type for each actor/task in the CSDF graph, where the energy consumption is minimized and throughput and latency constraints are met; 2) Then, based on the obtained processor type assignment, we propose a new approach to map actors of the streaming application specified as a CSDF graph to the given platform and then further reduce the energy consumption by using VFS. Experimental results show the effectiveness of our proposed algorithm in terms of energy efficiency in comparison to the related work. This contribution and experimental results are presented in Chapter 4.

Contribution 3: Novel Algorithm for Energy-Efficient Mapping of Real-Time Tasks on Heterogeneous Multicores Using Task Splitting

To address Problem 2 in Section 1.2, in the context of independent real-time tasks (applications), we are inspired by the latest C=D task-splitting technique [BDWZ12] and propose a novel algorithm, called *Allocation and Split on Heterogeneous Multicore systems* (ASHM), to energy-efficiently map real-time tasks on heterogeneous multicore systems by using the C=D task-splitting technique. To the best of our knowledge, our ASHM algorithm is the first work to consider the C=D task-splitting technique on heterogeneous multicore systems for energy efficiency. In this work, we investigate the application of the C=D task-splitting on real-time heterogeneous multicore systems to reduce energy consumption. The concepts regarding the task-splitting approach and the C=D approach will be explained later in Section 5.2.4. We analyze the properties of the C=D task-splitting and extend it for heterogeneous multicore systems. The analysis and extension of the C=D task-splitting on heterogeneous multicore systems serves as the foundation of the proposed ASHM algorithm. Experimental results show that our ASHM outperforms the existing approaches in terms of energy efficiency. This contribution and experimental results are presented in Chapter 5.

Contribution 4: The first Schedulability Test for Imprecise Mixed-Criticality Systems Under EDF-VD Scheduling

The last contribution in this dissertation addresses Problem 3 discussed in Section 1.2. We propose the first test to check the schedulability of the IMC model under the

scheduling and prove the correctness of the proposed schedulability test. A brief introduction of the EDF-VD scheduling algorithm is given in Section 6.2.3. Moreover, with the proposed schedulability test, a special metric for scheduling algorithms, namely *speedup factor*, is used to quantify the optimality of the IMC model under EDF-VD scheduling algorithm. The experimental results show that EDF-VD can schedule more IMC task sets than the existing approach AMC [BB13]. This contribution and its experimental results are presented in Chapter 6.

1.4 Dissertation Outline

The remainder of this dissertation is organized in a self-contained manner. **Chapter 2** introduces some common, fundamental, and relevant knowledge about the data-flow model, considered in this dissertation, and the real-time system models and their corresponding analysis techniques that are deployed in this dissertation are also presented in order to facilitate the understanding of the contributions afterwards.

Chapter 3 - 6 present in details the dissertation contributions briefly introduced in Section 1.3. Each chapter is organized in a self-contained way. That is, each chapter has its specific

- brief introduction and detailed contributions;
- related work and a system model;
- proposed algorithm/approach; and
- experimental results.

Finally, **Chapter 7** summarizes this dissertation and points out some directions which deserve further investigation. The detailed organization of this dissertation is as follows:

1. **Chapter 2** presents some common background information pertaining to the CSDF model, real-time systems, hard-real-time scheduling framework proposed in [BS11][BS12][BS13].
2. **Chapter 3** presents the new approach to optimize resource requirements of hard-real-time streaming applications subject to latency constraints in the scheduling framework proposed in [BS11][BS12][BS13].
3. **Chapter 4** presents the FDM algorithm to energy-efficiently map hard-real-time streaming applications onto cluster heterogeneous multicore systems.

4. **Chapter 5** presents the ASHM algorithm to energy-efficiently map real-time tasks onto heterogeneous multicore systems by using the task-splitting technique.

5. **Chapter 6** presents the schedulability test for the IMC model under EDF-VD and the proof of the speed-up factor.

6. **Chapter 7** summarizes the dissertation and discusses possible future work.

Chapter 2

Background

> Predictability, not speed, is the foremost goal in real-time system design.
>
> John Stankovic [Sta88]

In this chapter, to better understand this dissertation, we introduce some common preliminaries which we use in the subsequent chapters, such as the cyclo-static dataflow (CSDF) model, the real-time theories, and the hard-real-time scheduling of CSDF.

2.1 Cyclo-Static Dataflow (CSDF) Model

In this dissertation, we use the cyclo-static dataflow (CSDF) model to model streaming applications. In this section, we introduce this model and its properties.

In [BELP96], Bilsen *et al.* proposed the cyclo-static dataflow (CSDF) model to model signal processing applications. CSDF generalizes the well-known synchronous dataflow (SDF) model [LM87]. A CSDF graph is defined as a directed graph $G = (A, \mathcal{E})$, where A is a set of actors and $\mathcal{E}$ is a set of edges. Actor $\tau_j \in A$ represents a piece of computation in an application and edge $e_i \in \mathcal{E}$ represents the communication between two actors, where an atomic data object that is transferred via an edge is called a **token**. In a CSDF graph, every actor $\tau_i \in A$ has an **execution sequence** $[F_i(1), F_i(2), \cdots, F_i(\mathcal{N}_i)]$ of length $\mathcal{N}_i$, meaning that the nth execution/firing of actor τ_i executes the code of function $F_i(((n-1) \mod \mathcal{N}_i) + 1)$. Similarly, each CSDF actor may produce/consume a variable but predefined number of data tokens in consecutive executions, called **production/consumption sequence**. The *production/consumption sequence* has the same length of $\mathcal{N}_i$ as the *execution sequence*. An edge $e_u \in \mathcal{E}$ is a *first-in, first-out* (FIFO) queue defined as pair $e_u = (\tau_i, \tau_j)$, denot-

ing that actor τ_i produces data tokens on edge e_u and actor τ_j consumes data tokens from edge e_u. Let $[x_i^u(1), x_i^u(2), \cdots, x_i^u(\mathcal{N}_i)]$ denote the **production sequence** of actor τ_i on edge e_u, meaning that at the nth execution actor τ_i produces $x_i^u(((n-1) \mod \mathcal{N}_i)+1)$ data tokens on edge e_u. $X_i^u(n) = \sum_{l=1}^{n} x_i^u(l)$ denotes the total amount of data tokens which actor τ_i produces on edge e_u after its first n executions. Let $[y_j^u(1), y_j^u(2), \cdots, y_j^u(\mathcal{N}_j)]$ denote the **consumption sequence** of actor τ_j on edge e_u. Similarly, at the nth execution actor τ_j consumes $y_j^u(((n-1) \mod \mathcal{N}_j) + 1)$ data tokens from edge e_u and $Y_j^u(n) = \sum_{l=1}^{n} y_j^u(l)$ denotes the total amount of data tokens which actor τ_j consumes from edge e_u after its first n executions.

A compelling and important property of CSDF is its *decidability*, i.e., a schedule for all actors in a CSDF graph G can be derived at design time. To derive a valid static schedule of a CSDF graph at design-time, the graph needs to be **consistent** and **live**.

Definition 2.1.1 ([BELP96])**.** A CSDF graph G is said to be **consistent** if a non-trivial solution exists for a *repetition vector* $\vec{q} = [q_1, q_2, \cdots, q_{|A|}]^T$.

The *repetition vector* $\vec{q}$ is defined as follows:

Definition 2.1.2 ([BELP96])**.** Given a connected CSDF graph G, a vector $\vec{q} = [q_1, q_2, \cdots, q_{|A|}]^T$ representing the number of invocations of the actors of G in a valid static schedule is called a *repetition vector* of G.

And the *repetition vector* $\vec{q}$ is computed by using the following theorem,

Theorem 2.1.1 ([BELP96])**.** *For a connected CSDF graph G, a* repetition vector $\vec{q} = [q_1, q_2, \cdots, q_{|A|}]^T$ *is given by*

$$\vec{q} = \mathbf{\Theta} \cdot \vec{r}, \qquad \textit{with} \qquad \Theta_{ik} = \begin{cases} \mathcal{N}_i & \textit{if } i = k \\ 0 & \textit{otherwise} \end{cases} \tag{2.1}$$

where $\vec{r} = [r_1, r_2, \cdots, r_{|A|}]^T$ is a positive integer solution of the balance equation

$$O \cdot \vec{r} = \vec{0} \tag{2.2}$$

and where the topology matrix $O \in \mathbb{Z}^{|\mathcal{E}| \times |A|}$ is defined by

$$O_{uj} = \begin{cases} X_j^u(N_j) & \textit{if } \tau_j \textit{ produces on channel } e_u \\ -Y_j^u(N_j) & \textit{if } \tau_j \textit{ consumes from channel } e_u \\ 0 & \textit{Otherwise.} \end{cases} \tag{2.3}$$

Definition 2.1.3 ([BELP96])**.** A CSDF graph G is said to be **live** if a deadlock-free schedule can be found.

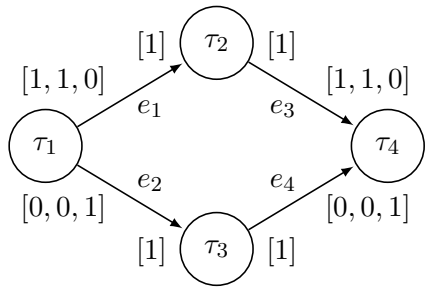

Figure 2.1: CSDF graph G

Definition 2.1.4. For a **consistent** and **live** CSDF graph, the graph completes one *iteration*, if every actor $\tau_i \in A$ executes for q_i times.

Below, we use an illustrative example to facilitate the understanding of the theories and definitions of CSDF presented above.

Example 2.1.1. Consider the CSDF depicted in Figure 2.1. There are four actors $\{\tau_1, \tau_2, \tau_3, \tau_4\}$ and four edges $\{e_1, e_2, e_3, e_4\}$. Each actor has different *production/ consumption sequences* on different edges. For example, actor τ_1 has a production sequence of $[1, 1, 0]$ on edge e_1 and actor τ_2 has a consumption sequence of $[1]$ on edge e_1 and a production sequence of $[1]$ on edge e_3. Then, according to Equation (2.1),(2.2), (2.3) in Theorem 2.1.1, we obtain

$$O = \begin{bmatrix} 2 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & -1 \end{bmatrix}, \vec{r} = \begin{bmatrix} 1 \\ 2 \\ 1 \\ 1 \end{bmatrix}, \mathbf{\Theta} = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}, \vec{q} = \begin{bmatrix} 3 \\ 2 \\ 1 \\ 3 \end{bmatrix}$$

In this dissertation, we consider streaming applications which are modeled as *acyclic* CSDF graphs. From empirical studies in [TA10], Thies and Amarasinghe showed that 90% of streaming applications can be modeled as acyclic SDF graphs, where SDF graphs are a subset of CSDF graphs. For acyclic CSDF graphs, we have the following lemma,

Lemma 2.1.1 ([BS11])**.** *Any acyclic consistent CSDF graph is live.*

2.2 Real-Time Theories

This section introduces the real-time task models, real-time scheduling algorithms, the schedulability analysis techniques and the multiprocessor real-time scheduling

algorithms.

2.2.1 Real-Time Task Models

A real-time system is comprised of a collection of real-time applications. To ensure the system's timing correctness, each application in the application set is modeled as a real-time task τ_i and all tasks form a real-time task set Γ. A real-time task τ_i that might generate an infinite sequence of task instances, also called *jobs*, is usually specified by a parameter tuple $\{S_i, C_i, D_i, T_i\}$ and the interpretations of the parameters are as follows:

- S_i denotes the start time of τ_i;
- C_i denotes the estimated *Worst-Case Execution Time* (WCET) of τ_i;
- D_i denotes the relative deadline of τ_i; and
- T_i denotes the period of τ_i or the minimal arrival interval of two subsequent jobs generated by τ_i.

The different interpretations of T_i define two widely-used real-time task models: *periodic task model* and *sporadic task model*. In the *periodic task model*, T_i denotes the period of task τ_i, i.e., task τ_i releases its jobs strictly periodically at time instants $t = \{t|S_i + kT_i, k \in \mathbb{Z}^+\}$. In the *sporadic task model*, T_i denotes the minimal arrival interval of two subsequent jobs generated by τ_i, i.e., task τ_i can release its next job anytime once the minimal arrival interval T_i elapses.

The *task model* also features another characeristic related to the tasks' start times. If all tasks in a task set Γ have the same start time, i.e., $S_1 = S_2 = \cdots = S_n$, task set Γ is said to be a *synchronous task set*. Otherwise, it is called an *asynchronous task set*. Moreover, considering the relation between deadline D_i and period T_i, a *periodic/sporadic* task set can be defined as either *implicit deadline* task set or *constrained deadline* task set. The *implicit deadline task set* indicates that every task $\tau_i \in \Gamma$ has $D_i = T_i$, whereas the *constrained deadline task set* means that every task $\tau_i \in \Gamma$ has $D_i \leq T_i$. In real-time systems, the *utilization* is used to denote the ratio of WCET over a period. For a task set Γ, the *utilization* u_i of task $\tau_i \in \Gamma$ is $u_i = C_i/T_i$ and thus the *total utilization* U_Γ of task set Γ is the sum of the utilizations of all tasks, i.e., $U_\Gamma = \sum_{i=1}^{n} u_i$. In addition, we have another term, called *density*, which denotes the ratio of task's WCET over the lesser of a task's relative deadline and period , i.e., the *density* δ_i of task $\tau_i \in \Gamma$ is $\delta_i = C_i/\min(D_i, T_i)$ and the *total density* δ_Γ of task set Γ is computed by $\delta_\Gamma = \sum_{i=1}^{n} \delta_i$.

Table 2.1 lists the corresponding task models used in different chapters. Note that in this section we only introduce the basic task model. The exact task model used

	Task Model		
	synchronous/ asynchronous	constrained/ implicit	periodic/ sporadic
Chapter 3	asynchronous	constrained deadline	periodic
Chapter 4	asynchronous	implicit deadline	periodic
Chapter 5	synchronous	constrained deadline	periodic
Chapter 6	synchronous	implicit deadline	sporadic

Table 2.1: Task models considered in the dissertation' chapters

in each chapter might be slightly different from the basic task model, and we will elaborate the difference in each chapter.

2.2.2 Real-Time Scheduling

When a system and a set of real-time tasks are given, a real-time scheduling algorithm is required to schedule the task set on the specified system. A scheduling algorithm schedules tasks based on their *priority* and there are three ways to assign *priority* to each task [DB11].

- **Fixed task priority**: all jobs from a task have a single fixed priority. An example of this is the *rate-monotonic scheduling* [LL73];
- **Task-level priority**: the jobs of a task might have different priorities, but each job has a single static priority. An example of this is the *earliest deadline first (EDF) scheduling* [LL73];
- **Dynamic priority**: a job of a task might have different priorities at different times. An example of this is the *least laxity first (LLF) scheduling* [Leu89].

Preemption is another important feature pertaining to scheduling algorithms.

- If tasks can be preempted by a task with higher priority at any time, the scheduling algorithm is said to be a *preemptive scheduling algorithm*;
- If tasks start to execute and they cannot be suspended at runtime until their completions, then the scheduling algorithm is said to be a *non-preemptive scheduling algorithm*.

Since the optimal scheduling algorithms for uniprocessor or multiprocessor systems are both *preemptive* [LL73][BCPV93a], we in this dissertation focus our study on *preemptive scheduling algorithms*.

Schedulability Tests and Scheduling Algorithms

As we discussed in Chapter 1, the key problem of real-time systems is to decide whether real-time tasks are schedulable on a specific platform under a certain real-time scheduling algorithm, i.e., to guarantee the system's timing constraint. *Schedulability tests* are the formal and veryfing tools to help us check the schedulability. The formal definition of *Schedulability tests* can be found in Definition 1.2.1 on page 9. In this section, we introduce the schedulability tests used in this dissertation.

Generally, schedulability tests can be classified as follows [DB11]:

- **Sufficient test**: If a real-time task set which is schedulable according to a schedulability test is indeed schedulable, the schedulability test is said to be *sufficient*.
- **Necessary test**: If a real-time task set which is unschedulable according to a schedulability test is indeed unschedulable, the schedulability test is said to be *necessary*.
- **Exact test**: If a schedulability test is both *sufficient* and *necessary*, the schedulability test is said to be *exact*.

In some cases, it is highly difficult to derive an exact test, so we would like to derive a **sufficient test** to ensure the schedulability of a real-time task set.

The *preemptive* earliest deadline first (EDF) scheduling algorithm [LL73] is the most studied dynamic-priority scheduling algorithm. In this dissertation, we use EDF as the scheduling algorithm in **Chapter 5** and **6**. The seminal paper [LL73] proved the *exact* schedulability test for an *implicit deadline periodic* task set under EDF on a uniprocessor system.

Theorem 2.2.1 ([LL73])**.** *For an implicit deadline periodic task set Γ, it is schedulable by EDF if and only if*

$$U_\Gamma = C_1/T_1 + C_2/T_2 + \cdots + C_n/T_n = \sum_{i=1}^{n} C_i/T_i \leq 1. \tag{2.4}$$

On a uniprocessor system, EDF has been proven to be the optimal scheduling algorithm for *implicit deadline tasks* [Der74].

Theorem 2.2.2 ([Der74])**.** *If a job set $\mathcal{J}$ is schedulable by an algorithm A, then it is schedulable by EDF.*

Corollary 2.2.1. *EDF is an optimal scheduling algorithm on a uniprocessor system.*

However, for *constrained deadline* task set, Equation (2.4) only serves as a *necessary* test. Barauh *et al.* in [BMR90] proposed an *exact* schedulability test for *constrained deadline* task set under EDF on a uniprocessor system, where the *exact* schedulability test is derived based on the concept of *demand bound function* (dbf).

$$\mathsf{dbf}(\tau_i, t_0, t_f) = \max\left\{0, \left\lfloor \frac{(t_f - t_0) - D_i}{T_i} \right\rfloor + 1\right\} C_i \tag{2.5}$$

where $t_f - t_0$ denotes a time interval, t_0 and t_f are the start time and the end time of the interval, respectively. dbf computes the maximum cumulative execution time which a task demands within time interval $[t_0, t_f]$. If the total maximum execution time of task set Γ within the time interval does not exceed the time interval, the task set is schedulable. Otherwise, it is unschedulable for the task set.

Theorem 2.2.3 ([BMR90])**.** *A task set Γ is schedulable if and only if $U_\Gamma \leq 1$ and*

$$\forall t < L_a, \quad \mathit{dbf}(\Gamma, t_0, t_f) = \sum_{i=1}^{n} \max\left\{0, \left\lfloor \frac{(t_f - t_0) - D_i}{T_i} \right\rfloor + 1\right\} C_i \tag{2.6}$$

where L_a is defined as follows:

$$L_a = \max\left\{D_1, \cdots, D_n, \max_{1 \leq i \leq n}\{T_i - D_i\} \frac{U_\Gamma}{1 - U_\Gamma}\right\} \tag{2.7}$$

Theorem 2.2.3 is the exact test to check the schedulability of a task set under EDF scheduling. However, the exact test shown in Theorem 2.2.3 is computationally expensive because this exact test needs to test all absolute deadlines within the time interval and there can be a large number of absolute deadlines which need to be checked. To improve the efficacy of the EDF exact test, Zhang and Burns [ZB09] proposed a new exact test for the EDF scheduling, referred as Quick convergence Processor-demand Analysis (QPA).

Theorem 2.2.4 ([ZB09])**.** *A task set Γ is schedulable if and only if $U_\Gamma \leq 1$ and the result of the QPA iterative algorithm shown in Algorithm 1 is $\mathit{dbf}(\Gamma, t_o, t_f) \leq d_{\min}$, where $d_{\min} = \min\{D_i\}$.*

The extensive experimental results in [ZB09] demonstrate the efficiency of QPA in terms of reducing the time complexity of testing the schedulability. Therefore, in our work, we use QPA to test the schedulability of a task set when the utilization-based test is not applicable.

Algorithm 1: QPA

```
1 t ← max{d_i | d_i < L};
2 while (dbf(Γ, t) ≤ t ∧ dbf(Γ, t) > d_min)
3 {if (dbf(Γ, t) < t) t ← dbf(Γ, t);
4 else t ← max{d_i | d_i < L};
5 }
```

2.2.3 Multiprocessor Real-Time Scheduling

Nowadays, the increasing number of real-time systems is implemented on multiprocessor platforms. The prevalence of these real-time multiprocessor systems rises a new problem, namely the *assignment problem*, i.e., deciding which processor to execute which task. Multiprocessor scheduling algorithms can be classified as follows based on the tasks' assignment:

- *Partitioned scheduling*: tasks are only permitted to execute on their assigned processors and task migration is prohibited. On each processor, a uniprocessor scheduling algorithm is deployed to schedule the tasks assigned to the processor;

- *Global scheduling*: tasks are permitted to migrate to any processor at anytime and a global scheduling algorithm assigns tasks to proper processors at runtime;

- *Cluster Scheduling:* the system is comprised of several clusters where each cluster consists of a number of processors. Tasks are statically assigned to a fixed cluster and within a cluster tasks are permitted to migrate to the processors in the same cluster, i.e., a global scheduling algorithm is deployed within a cluster, but task migration between clusters is prohibited;

- *Semi-partitioned scheduling/Task-splitting*: the majority of tasks are statically assigned to processors and only a few tasks are permitted to migrate among processors. Usually, the assignment of migrative tasks is also known at design time, i.e., a migrative task executes partially on one processor and then migrates to another processor to complete its execution.

Assignment Algorithms

When *partition scheduling* is deployed to schedule tasks on a multiprocessor system, a key problem is to efficiently decide how to *assign* a task to a proper processor so

that a certain metric, e.g., schedulability, energy-efficiency, etc, is satisfied[1].

The *assignment problem* of real-time tasks on a multiprocessor system is inherently analogous to the well-known *bin-packing* problem [GJ79]. In the **bin-packing** problem, objects of different volumes are packed into a finite number of bins with fixed capacity such that the number of bins used is minimized. The *bin-packing* problem has been proven to be a **NP**-complete problem [GJ79], so an optimal solution cannot be obtained in a polynominal time unless **P=NP**. Therefore, many heuristic algorithms are developed to efficiently solve the *bin-packing* problem and to obtain a suboptimal result in a reasonable time. The close similarity between these two problems allows us to directly utilize the well-established heuristic algorithms for the *bin-packing* problem to assign real-time tasks on a multiprocessor system. Below, we introduce the most used heuristic algorithms [CGJ97, Joh74].

- **First-Fit (FF)** algorithm: the **FF** algorithm always tries to place an item I_i to the first bin B_j (i.e., lowest index). That is

  $$j = \min\{k : \text{size}(I_i) + \text{capacity}(B_k) \leq 1\}$$

  If no exsiting bin can accomodate the item, a new bin is opened and the item is placed in the new bin;

- **Worst-Fit (WF)** algorithm: the **WF** algorithm always tries to assign an item I_i to the bin B_j which has the most residual capacity after placing item I_i. That is

  $$j = \min\{k : \text{size}(I_i) + \text{capacity}(B_k) \text{ minimized}\}$$

  If no exsiting bin can accomodate the item, a new bin is opened and the item is placed in the new bin;

- **Best-Fit (BF)** algorithm: the **BF** algorithm always tries to assign an item I_i to the bin B_j which has the least residual capacity after placing item I_i. That is

  $$j = \min\{k : \text{size}(I_i)+\text{capacity}(B_k) \text{ maximized \& not exceed the bin volume.}\}$$

  If no exsiting bin can accomodate the item, a new bin is opened and the item is placed in the new bin.

Performance of these heuristic algorithms can be improved by sorting tasks in order of decreasing utilization or density. Then, we have:

[1]Throughout this dissertation, we may use the term "assign", "map", and "partition" interchangeably to denote the procedure that statically decides a processor for a task to complete its execution.

	Priority	Platform	Multiprocessor Scheduling
Chapter 3	task-level/dynamic	multiprocessor	global scheduling
Chapter 4	dynamic	multiprocessor	cluster scheduling
Chapter 5	task-level	multiprocessor	task-splitting
Chapter 6	dynamic [2]	uniprocessor	None

Table 2.2: Scheduling algorithms considered in each chapter

- **First-Fit-Decreasing (FFD)**: all tasks are sorted in decreasing order of their utilization or density (see Section 2.2.1) and then tasks are assigned using the **FF** algorithm;

- **Worst-Fit-Decreasing (WFD)**: all tasks are sorted in decreasing order of their utilization or density and then tasks are assigned using the **WF** algorithm;

Although **partitioned scheduling** has no migration cost and can directly apply a wealth of well-developed uniprocessor real-time theories, it suffers from low resource utilization due to the capacity loss during the assignment procedure [DB11]. The rest of the multiprocessor real-time scheduling algorithms can achieve better resource utilization and additionally the research on them is an increasingly hot topic in the real-time community. Therefore, in this dissertation, we consider the *global scheduling, cluster scheduling, and task-splitting approach*.

In the *cluster scheduling*, the inital step is to assign tasks to a cluster such that a global scheduling can be applied to the tasks assigned to the cluster. Similarly, in the *task-splitting approach*, a subset of tasks are statically assigned to processors and the rest of the tasks are splitted among the processors. The assignment procedures in both the *cluster scheduling* and the *task-splitting aproach* are similar to the assignment procedure in *partitioned scheduling*. Therefore, all heuristic assignment algorithms introduced aboven can be applied to the *cluster scheduling* and the *task-splitting approach*.

Table 2.2 summarizes the priority assignment schemes, platforms, and multiprocessor scheduling algorithms considered in each chapter.

[2] Although the original EDF is a task-level priority scheduling algorithm, EDF-VD may change deadlines of some tasks during runtime. As a result, the priority of these tasks will be changed upon their execution. Thus, we here consider EDF-VD as a dynamic priority scheduling.

2.3 Hard-Real-Time (HRT) Scheduling of CSDF graphs

Throughout this dissertation, instead of traditional dataflow scheduling techniques [MB07], we use a new scheduling framework [BS11, BS12, BS13] to schedule CSDF graphs. In this section, we give a brief introduction about this new scheduling framework.

Traditionally, CSDF graphs are scheduled by self-timed scheduling [MB07], where an actor starts to execute as soon as it receives enough tokens from its predecessors. However, since the self-timed scheduling normally provides best-effort services, it is difficult to provide a hard-real-time timing guarantee for every task in an application. Bamakhrama and Stefanov in [BS11, BS12, BS13] proposed a Hard-Real-Time (HRT) scheduling framework in which an acyclic CSDF graph is converted into an independent periodic task set. This conversion effectively bridges the gap between data-flow models and real-time theories and enables us to apply a plethora of well-developed real-time theories to CSDF graphs. The advantage of this framework is the direct application of real-time theories on dataflow models, such as schedulability tests and assignment algorithms, and the designers are able to accomplish *fast admission control* and *temporal isolation* and provide *hard-real-time guarantees*. A good example of the application of the *HRT* scheduling framework is demonstrated in [BZNS12], where the merit of the *HRT* scheduling framework helps to significantly reduce the complexity of the *design space exploration* when designing a real-time streaming multiprocessor system.

The *HRT* scheduling framework takes as an input a CSDF graph where the WCET of each actor in the CSDF graph is known in a priori, and finally it outputs a periodic task sets which can be scheduled by a real-time scheduler. **Note** that the WCETs considered in the *HRT* scheduling framework also account for the **worst-case communication overhead** because the *HRT* scheduling framework strives to ensure the feasibility of this approach regardless of the variance of different task assignments. The basic concept behind this framework is to derive the real-time parameters, i.e., period, start time, and deadline, for each actor according to actors' WCETs and the CSDF graph properties, e.g., the repetition vector explained in Section 2.1. The procedure goes through the following steps: **1)** it computes a period for each actor in the input CSDF graph according to its *repetition values* and WCETs; **2)** it computes the actors' *start times* such that the precedence constraints between actors are respected and thus the deadlock in execution is avoided; and **3)** it determines the *deadline* of each actor.

Computing Periods

To compute the period of an actor in a CSDF graph, we first define the *workload* of an actor and *the maximum actor workload* of a graph as follows:

Definition 2.3.1. The **workload** of an actor τ_i is $W_i = q_i C_i$ and the **maximum actor workload** of the graph is $\hat{W} = \max_{\tau_i \in G}\{W_i\}$

Then, we can use the *maximum actor workload* and the *repetition value* q_i of actor τ_i to compute the minimum period $\breve{T}_i$ of actor τ_i as follows [BS11]:

$$\breve{T}_i = \frac{\text{lcm}(\vec{q})}{q_i} \left\lceil \frac{\hat{W}}{\text{lcm}(\vec{q})} \right\rceil \tag{2.8}$$

where $\text{lcm}(\vec{q})$ is the *least common multiple* of the *repetition vector* $\vec{q}$ (explained in Section 2.1). The minimum periods deliver the maximal throughput for the CSDF graph under the *HRT* scheduling. Based on this minimum period, a *period scaling technique* can be applied to uniformly scale up the period of each actor under the *HRT* scheduling framework, thereby adjusting the throughput of the CSDF graph [ZBS13, SLS16] .

Computing Start Times

Once the periods of all actors are computed, we need to deal with the precedence constraints between actors. The execution semantics of a dataflow model requires an actor to receive sufficient data from its predecessors in order to trigger its execution, thus the existence of precedence constraints prohibits all actors to start their execution at the same time. To resolve the precedence constraints, the converting procedure in the *HRT* scheduling framework offsets the start times of actors such that by its start time every actor is capable of obtaining sufficient data on its input edges and its subsequently periodic executions are free from deadlock. The following lemma is used to compute the earliest start time for each actor in a CSDF graph

Lemma 2.3.1 (From [BS11])**.** *For an acyclic CSDF graph G, the earliest start time of an actor $\tau_j \in A$, denoted S_j, under HRT scheduling is given by*

$$S_j = \begin{cases} 0 & \text{if } \Omega(\tau_j) = \emptyset \\ \max_{\tau_i \in \Omega(\tau_j)} (S_{i \to j}) & \text{if } \Omega(\tau_j) \neq \emptyset \end{cases} \tag{2.9}$$

where $\Omega(\tau_j)$ is the set of predecessors of τ_j, and $S_{i \to j}$ is given by

$$S_{i \to j} = \min_{t \in [0, S_i + \alpha]} \{t : \underset{[S_i, \max(S_i, t) + k)}{\mathsf{prd}} (\tau_i) \geq \underset{[t, \max(S_i, t) + k]}{\mathsf{cns}} (\tau_j) \; \forall k \in [0, \alpha]\} \tag{2.10}$$

where $\alpha = q_i T_i = q_j T_j$, $\mathsf{prd}_{[t_s, t_e)}(\tau_i)$ is the number of tokens produced by τ_i during the time interval $[t_s, t_e)$, and $\mathsf{cns}_{[t_s, t_e]}(\tau_j)$ is the number of tokens consumed by τ_j during the time interval $[t_s, t_e]$.

Determing Deadlines

After the step of computing the start times, we have three parameters to specify an actor τ_i as a periodic task: the given WCET C_i, the period T_i, and the start time S_i computed by using Equation (2.8) and Equation (2.9), respectively. Recall that a periodic task is specified by a tuple of 4 parameters $\{S_i, C_i, D_i, T_i\}$, so we need to determine a deadline for each actor. In the *HRT* scheduling framework, the deadline D_i of actor τ_i can be selected within a well-defined range $[C_i, T_i]$, i.e., $D_i \in [C_i, T_i]$. The deadline selection is a highly relevant problem for the *HRT* scheduling framework because the deadline selection is able to influence both the performance (mainly the latency) of the CSDF graph and the number of processors required to schedule the CSDF graph. Later in Chapter 3, we propose a novel approach to optimally select deadlines in the *HRT* scheduling framework such that the latency requirement is ensured and the number of processors required is minimized.

Latency and Throughput

After the periodic tasks' parameters are computed, as explained above, the latency and throughput of the CSDF graph scheduled in the *HRT* scheduling framework can be computed. Equation (2.11) is used to compute the minimum latency of the CSDF graph,

$$L(G) = \max_{w \in \mathbf{W}} (S_{\text{out}} + (g^C_{\text{out}} + 1)T_{\text{out}} - (S_{\text{in}} + g^P_{\text{in}} T_{\text{in}})) \tag{2.11}$$

where w is one path of set $\mathbf{W}$ which consists of all paths from the input actor to the output actor. Here, S_{out} and T_{out} are the start time and period, respectively, of output actor τ_{out}, while S_{in} and T_{in} denote the start time and period, respectively, of input actor τ_{in}. g^C_{out} and g^P_{in} are two constants which denote the number of invocations the actor waits for the non-zero consumption/production of tokens on a path $w \in \mathbf{W}$. Note that when $D_i = C_i, \forall i$, the graph can reach the minimum latency achievable by the *HRT* scheduling framework. The throughput of the CSDF graph is computed as follows:

$$\mathcal{R} = 1/T_{\text{out}} \tag{2.12}$$

Note that when all actors have the minimum periods $\breve{T}_i$, the graph can reach the maximum throughput achievable by the *HRT* scheduling framework.

Chapter 3

Resource Optimization for Real-Time Streaming Application

Di Liu, Jelena Spasic, Jiali Teddy Zhai, Gang Chen, and Todor Stefanov,
"Resource optimization for CSDF-modeled streaming applications with latency constraints,"
2014 *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, 2014, pp. 1-6.

STREAMING applications, such as video/audio processing and digital signal processing, contain ample amount of parallelism which perfectly matches the processing power of Multi-Processor System-on-Chip (MPSoC) platforms. To efficiently program MPSoC platforms, Models-of-Computation (MoCs) are used to specify streaming applications. Prominent examples of MoCs include Synchronous Data Flow (SDF) [LM87] and its generalization Cyclo-Static Dataflow (CSDF) [BELP96].

Traditionally, CSDF graphs are scheduled by self-timed scheduling [MB07], where an actor starts to execute as soon as it receives enough tokens from its predecessors. However, since the self-timed scheduling normally provides best-effort services, it is difficult to provide a hard-real-time timing guarantee for every actor in a CSDF graph. Bamakhrama and Stefanov in [BS11] proposed the hard-real-time (HRT) scheduling framework to schedule acyclic CSDF graphs, explained in detail in Section 2.3. This HRT scheduling framework provides *HRT timing guarantee* and *fast admission control* for an application modeled as a CSDF graph, at the expense of increasing the graph latency [BS12]. The same authors in [BS12] identified this issue of the HRT scheduling framework and proposed to reduce the graph latency by scaling down actors' relative deadlines. In [BS12], Bamakhrama and Stefanov proposed to use a scaling factor to uniformly reduce the deadlines of all actors/tasks. However, since the deadlines of periodic tasks play a crucial role in determining the minimum number of

processors required to schedule the task set, this uniform scaling factor may unnecessarily increase the required number of processors.

In this chapter, we address this above-mentioned problem of minimizing the number of processors required to schedule a latency-constrained streaming application modeled as a CSDF graph, which is scheduled by using the HRT scheduling framework. We formalize this problem and prove that it is an integer convex programming problem such that it can be solved effectivelly by an off-the-shelf convex programming solver, e.g., CVX [GB14]. The novel contributions of our work can be summarized as follows:

- We present a new method to compute the earliest starting time of actors in a CSDF graph when the actors are scheduled under the HRT scheduling.
- Based on the above contribution, we formalize the problem of minimizing the number of processors for a latency-constrained CSDF graph under the *HRT* scheduling and prove that it is an integer convex programming (ICP) problem.
- We carry out experiments by solving the ICP problem on 13 real-life streaming applications and demonstrate the effectiveness and efficiency of our solution approach in comparison to the deadline selection approach [BS12] in terms of the minimum number of processors required to schedule an application. By applying our approach, we obtain reduction in the number of processors in more than 48% of the conducted experiments.

Spasic *et al.* in [SLCS16] proposed a new approach to improve the HRT scheduling framework. The proposed approach in this chapter is also applicable to the improved HRT scheduling and more details can be found in [SLCS16].

3.1 Background

We have introduced the CSDF application model, real-time theories, and the HRT scheduling framework in Chapter 2. Here, we present the system model considered in this chapter. The platform, we target, is a homogeneous multiprocessor platform which consists of identical processors. Since the global scheduling, explained in Section 2.2.3, has been proven to be a theoretically optimal scheduling algorithm on multiprocessor systems [DB11], we adopt a global optimal scheduling, such as PFair [BCPV93b], to schedule periodic tasks generated from an acyclic CSDF graph.

To reduce the graph latency, we scale down the relative deadline of each actor and thus a *constrained deadline* task set (i.e., deadline is smaller than or equal to period) is derived. For the *constrained deadline* task model (i.e., $D_i \leq T_i$), Baker and Baruah in

[BB07] gave the following sufficient schedulability for the global optimal scheduling:

$$\delta_\Gamma \leq M \tag{3.1}$$

δ_Γ is the total density of a real-time task set Γ, explained in Section 2.2.1. M denotes the number of processors. This sufficient test can be converted to compute the minimum required number of processors for the global optimal scheduling as follows:

$$M = \lceil \delta_\Gamma \rceil \tag{3.2}$$

For global optimal scheduling, we see from Equation (3.2) that the total density δ_Γ plays a crucial role in computing the minimum number of processors needed to schedule a task set. Therefore, *we are able to minimize the number of processors required to schedule a task set* Γ *by minimizing the value of the total density* δ_Γ.

3.2 Related Work

In the context of real-time systems, several works deal with period and/or deadline selection for periodic tasks in order to achieve certain goals. The authors in [DZDN$^+$07] optimize periods for dependent tasks on hard real-time distributed automotive systems in order to meet a latency constraint. In [HCH11], Hong et al. propose a distributed approach to assign local deadlines for each task on distributed systems to meet a latency constraint. In contrast to [DZDN$^+$07] and [HCH11], our work selects deadlines for data dependent tasks in order to meet a latency constraint while minimizing the number of processors required for scheduling the application. Such minimization is not considered in [DZDN$^+$07] and [HCH11]. Balbastre et al. [BRC06] propose an analysis to select deadlines for periodic tasks on a uniprocessor to reduce the output jitters. Comparing to [BRC06], our work differs in that we select deadlines in order to reduce the required number of processors in a multiprocessor system while guaranteeing the latency constraint. Chantem et al. [CWLH08] optimize the periods and deadlines simultaneously for an infeasible independent task set such that it can be scheduled on a uniprocessor. Their work concentrates on the schedulability of a system rather than optimizing the resources while meeting the latency constraint which is the main goal of our work.

In another aspect, only a few works deal with latency of streaming applications specified as dataflow/task graphs. Given latency or throughput constraints, Javaid et al. [JHIP10] optimized the area of MPSoCs which are comprised of Application Specific Instruction set Processors (ASIP). The problem is formulated as an integer linear programming (ILP) problem. In their work, the area is optimized by setting different configurations for each ASIP. In contrast to their work, we consider to minimize the

number of processors and our objective function cannot be written in a linear form and thus not amenable to an ILP formulation. The authors in [CGHJ09] proposed a framework to synthesize homogeneous multiprocessor system for streaming applications with throughput constraints while optimizing latency and resources. However, their framework can not take the latency as a constraint. As a result, the framework in [CGHJ09] is not applicable to our problem. It is worth noting that the throughput constraint in [JHIP10] and [CGHJ09] can be trivially added into our approach.

3.3 Motivational Example

In this section, we take the CSDF graph in Figure 2.1 as our motivational example to demonstrate the deficiency of the approach in [BS12], where deadlines of actors are computed by Equation (3.3) below with the global uniform deadline scaling factor df.

$$\forall \tau_j \qquad D_j = C_j + df \times (T_j - C_j) \qquad 0 \leq df \leq 1 \tag{3.3}$$

We use Equation (3.2) to compute the required number of processors. Given a latency constraint of 20 clock cycles, if we use the approach in [BS12], it finds that the global deadline scaling factor df should be set to 0 in order to meet the latency constraint. By using Equation (3.3), we compute that $D_j = C_j$, and the parameters of the tasks are given in Table 3.1. Using these parameters we obtain that the total density $\delta_A = \frac{2}{2} + \frac{3}{3} + \frac{3}{3} + \frac{6}{6} = 4$. This means that 4 processors are needed to schedule the task set.

However, larger deadlines can be selected for some actors without violating the latency constraint, thereby reducing the total density δ_A, which in turn can decrease the number of processors. We select new deadlines $D_2 = 9$ and $D_3 = 12$ for actors τ_2 and τ_3, respectively, and recompute the start time of the tasks using Lemma 2.3.1 in Section 2.3. We see that in this specific case shown in Table 3.2, although we have changed two deadlines, the start times S_j have not changed. By using Equation (2.11) in Section 2.3 to compute the latency, we see that the latency of 20 clock cycles can be met with the new parameters, but the total density $\delta_A = \frac{2}{2} + \frac{3}{9} + \frac{3}{12} + \frac{6}{6} = 2.58$ decreases. This means that 3 processors are sufficient to schedule the task set without violating the latency constraint of 20 clock cycles. We can see from the motivational example that the approach from [BS12] is not optimal in terms of the required number of processors.

3.4 Proposed Approach

As we show in Section 3.3, although the deadline selection approach in [BS12] is able to meet the latency constraint, it is not optimal in terms of the number of processors. Hence, selecting deadlines in a proper way is a problem that should be solved in order

task	S_j	C_j	D_j	T_j
τ_1	0	2	2	6
τ_2	2	3	3	9
τ_3	14	3	3	18
τ_4	14	6	6	6

Table 3.1: Tasks Parameters 1

task	S_j	C_j	D_j	T_j
τ_1	0	2	2	6
τ_2	2	3	**9**	9
τ_3	14	3	**12**	18
τ_4	14	6	6	6

Table 3.2: Tasks Parameters 2

to minimize the number of processors while meeting the latency constraint. To select deadlines properly, we devise the solution approach presented in this section that formalizes and formulates the problem as a mathematical programming problem.

According to Equation (2.11), the latency depends on the earliest start time and deadline of the output actor, and the earliest start time of the input actor. The earliest start time S_j of any actor depends on two conditions: 1) at the earliest start time, there should be sufficient number of tokens on all input edges to enable the actor's firing and 2) once an actor fires for the first firing, the consequent firings of the actor should be possible to happen at time instant $t = S_j + kT_j$ for each $k \in \mathbb{N}^+$. The first condition is imposed by the firing rule [BELP96] of the CSDF model which is a data-driven model, where a sufficient number of tokens is the requirement to trigger an actor firing. The second condition makes sure that the CSDF graph is schedulable as a periodic task set. Although Lemma 2.3.1 in Section 2.3 is able to find the earliest start time of an actor, it is impossible to use its equations into any mathematical programming problem. Hence, we present a new computation method to calculate the start time of actors in a CSDF, in which the start times of actors can be represented as linear items and can be integrated into a mathematical programming problem.

Lemma 3.4.1. *For an acyclic CSDF graph G, the earliest start time of an actor $\tau_j \in A$, denoted S_j, under HRT schedule is given by*

$$S_j = \begin{cases} 0 & \text{if } \Omega(\tau_j) = \emptyset \\ \max_{\tau_i \in \Omega(\tau_j)} \{S_i + (S_{i \to j}^{min} - S_i^{min} - C_i) + D_i\} & \text{if } \Omega(\tau_j) \neq \emptyset \end{cases} \tag{3.4}$$

where $\Omega(\tau_j)$ is the set of predecessors of τ_j, S_i, C_i, and D_i are the earliest start time, WCET, and deadline of the predecessor actor τ_i, respectively. S_i^{min} is the earliest start time of τ_i given by Equation (2.9) when $D_n = C_n, \forall \tau_n \in A$, and $S_{i \to j}^{min}$ is given by Equation (2.10) when $D_n = C_n, \forall \tau_n \in A$.

Proof. Consider an arbitrary edge $e_u = (\tau_i, \tau_j) \in \mathcal{E}$. τ_j starts after τ_i has started and fired a "certain" number of times. This number of firings is independent from the execution speed of the actors and depends only on the production and consumption

rates of τ_i and τ_j on e_u. The production and consumption functions are given by:

$$\underset{[t_s,t)}{\mathsf{prd}}(\tau_i) = \sum_{k=0}^{\lfloor (t-t_s)/T_i \rfloor} (x_i^u(((k-1) \bmod \mathcal{N}_i) + 1) \cdot u(t - kT_i - D_i))$$

$$\underset{[t_s,t]}{\mathsf{cns}}(\tau_j) = \sum_{k=0}^{\lceil (t-t_s)/T_j \rceil} (y_i^u(((k-1) \bmod \mathcal{N}_j) + 1) \cdot u(t - kT_j))$$

where $x_i^u(k)$ is the k^{th} element in the production sequence of actor τ_i, $y_j^u(k)$ is the k^{th} element in the consumption sequence of actor τ_j, $\mathcal{N}_i$ and $\mathcal{N}_j$ are the execution lengths of τ_i and τ_j, respectively, as defined in [BELP96]. $u(t)$ is the unit step function. Suppose that $D_n = C_n, \forall \tau_n \in A$. The curves that depict the production and consumption functions of τ_i and τ_j are plotted in Figure 3.1. Interval Δ in Figure 3.1 depends only on the production and consumption rates of τ_i and τ_j on e_u and can be calculated as:

$$\Delta = S_{i \to j}^{\min} - S_i^{\min} - C_i \tag{3.5}$$

Now, suppose that $D_n > C_n, \forall \tau_n \in A$. The production curve will move to the right for certain time units, and the new start time of τ_i is S_i. If the consumption curve does not move, the relation between the production and consumption given by Equation (2.10) will be violated, i.e. it will happen in some point in time that the cumulative consumption is greater than the cumulative production. This means that we have to move the consumption curve to the right by the same number of time units such that the new start time $S_{i \to j}$ is minimum and the relation is preserved. Because the production and consumption rates are unchanged, interval Δ will stay the same, and we can calculate it as follows:

$$\Delta = S_{i \to j} - S_i - D_i \tag{3.6}$$

We can re-write Equation (3.5) and Equation (3.6) as:

$$S_{i \to j} = S_i + (S_{i \to j}^{\min} - S_i^{\min} - C_i) + D_i \tag{3.7}$$

□

Now, we can derive from Equation (3.4) the following set of linear inequality constraints, where the number of the linear inequality constraints is equal to the number of edges in the CSDF:

$$S_i + (S_{i \to j}^{\min} - S_i^{\min} - C_i) + D_i \leq S_j \quad \forall e_u \in \mathcal{E} \tag{3.8}$$

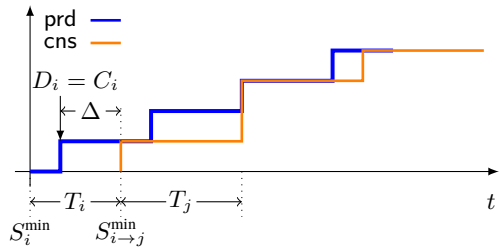

Figure 3.1: Production and consumption curves on edge $e_u = (\tau_i, \tau_j)$

As we explain Since computing the required number of processors depends on the total density δ_A of the task set (see Equation 3.2 Section 3.1), for a CSDF graph $G = (A, \mathcal{E})$, our objective is to minimize δ_A in order to minimize the number of processors. Therefore, we formulate our density minimization (DM) problem as follows:

$$\text{Minimize} \quad \delta_A = \sum_{\tau_n \in A} \frac{C_n}{D_n} \tag{3.9a}$$

$$\text{subject to:} \quad S_{\text{out}} + D_{\text{out}} - S_{\text{in}} \leq L + g_{\text{in}}^P T_{\text{in}} - g_{\text{out}}^C T_{\text{out}} \quad \forall w_{\text{in}\to\text{out}} \in W \tag{3.9b}$$

$$S_i + D_i - S_j \leq -(S_{i\to j}^{\min} - S_i^{\min} - C_i) \quad \forall e_u \in E \tag{3.9c}$$

$$-D_n \leq -C_n, D_n \leq T_n \quad \forall \tau_n \in A \tag{3.9d}$$

where Equation (3.9a) is the objective function and D_n is an optimization variable. We want the objective function (3.9a) with $|A|$ optimization variables to be subject to a latency constraint L. Therefore, Inequality (3.9b) comes from Equation (2.11). In addition, inequality constraints (3.9c) are the constraints given by (3.8), and inequality (3.9d) bounds all optimization variables in the objective function by the worst-case execution time and period as explained in Section 2.3. S_i and S_j (including S_{in}, S_{out}) are implicit variables which are not in the objective function (3.9a), but still need to be considered in the optimization procedure. L, $g_{\text{in}}^P T_{\text{in}}$, $g_{\text{out}}^C T_{\text{out}}$, $S_{i\to j}^{\min}$, $S_i^{\min}$, C_n, and T_n are constants.

Theorem 3.4.1. *The DM problem (3.9) is an integer convex programming (ICP) problem.*

Proof. First, we prove that the DM problem is a convex programming problem if the values of D and S are continuous. In a convex programming problem, the objective function and the constraints both should be convex[BV04]. We first prove the convexity of the objective function.

$$f(x) = \frac{a}{x} \tag{3.10}$$

Function (3.10) has been proven to be convex for $x \in (0, \infty)$ and $a > 0$ [BV04]. Since D_n is always greater than 0, all $\delta_n(D_n) = \frac{C_n}{D_n}$ are convex functions, where D_n and C_n are the variable x and the constant a, respectively. Moreover, if f_1 and f_2 are both convex, so is their sum $f_1 + f_2$. Hence, $\delta_A = \sum_{\tau_n \in A} \frac{C_n}{D_n}$ is a convex function.

A closed halfspace which is convex is a set of the form $\{\mathbf{x}|\mathbf{a}^T\mathbf{x} \leq \mathbf{b}\}$[BV04], where $\mathbf{a} \neq \mathbf{0}$ and all entries in $\mathbf{x}$ are continuous. Since all constraints (3.9b), (3.9c), and (3.9d) are in the form of the closed halfspace, all constraints are convex. Hence, the DM problem (3.9) is a convex programming problem.

Given that all D and S in the DM problem (3.9) have to take only integer values in practice, the DM problem is an ICP problem. □

In mathematical programming, a convex programming problem can be solved efficiently to find a global optimum. If the variables have to only take integer values, the problem becomes an integer convex programming problem. This integer problem is an NP-hard problem that can not be solved efficiently using only the conventional convex programming. Fortunately, the combination of the conventional convex programming [BV04] and some algorithms for solving mixed integer linear programming can be used to find a global optimal solution for ICP. In Section 3.5.2, the evaluation shows the efficiency of the existing CVX solver [GB14] to solve our DM problem.

3.5 Evaluation

In this section, we evaluate our DM approach and compare it with the Baseline Approach (BA) proposed in [BS12]. This baseline approach uses Binary Search to find the maximum df (in Equation 3.3) which makes the latency constraint met. Finding this maximum df reduces the required number of processors to schedule the CSDF actors. Our DM problem is solved by using mixed integer disciplined convex programming (MIDCP) in CVX [GB14]. All experiments are performed on an Intel i7 dual-core processor running at 2.7GHz with 4 GB RAM.

We have selected 13 real-life streaming applications modeled as CSDF graphs from the StreamIt [TA10] benchmark suit. The WCET of each actor in the application benchmarks which we use in this evaluation is the same as specified in [BS11]. The characteristics of these benchmarks are given in Table 3.3, including the number of actors ($|A|$), the number of edges ($|\mathcal{E}|$), the maximum latency (L_{max}) and the minimum latency (L_{min}) in clock cycles. The maximum latency is the latency obtained by using the implicit deadline periodic task model, i.e. $df = 1$, whereas the minimum latency is the latency obtained when $df = 0$. To demonstrate the effectiveness of our DM approach, the latency constraints of the graphs are varied during the experiments. We evaluate our DM approach in terms of the number of processors needed for each

Realistic Applications	$\|A\|$	$\|\mathcal{E}\|$	$L_{\max}$	$L_{\min}$
Beamformer	57	70	60912	14692
ChannelVocoder	55	70	284000	106755
DCT	8	7	380928	121672
Data Encryption Standard (DES)	53	60	46080	15602
FilterBank	85	99	158368	34638
MPEG2	23	26	138240	49452
Serpent	120	128	370296	122108
Time Delay Equalization (TDE)	29	28	1071840	628151
Vocoder	114	147	291360	21554
CD2DAT	6	5	829	258
H.263	4	3	996697	369508
Samplerate	6	5	3792	1531
Satelite	22	26	11746	5484

Table 3.3: Characteristics of application benchmarks

Constraint	Latency
L_0	$L_{\min}$
L_1	$0.4(L_{\max} - L_{\min}) + L_{\min}$
L_2	$0.9(L_{\max} - L_{\min}) + L_{\min}$

Table 3.4: Latency Constraints

benchmark and compare it to the BA for the three latency constraints per benchmark, shown in Table 3.4, while the achieved throughput of each benchmark is the same in both BA and DM approaches.

3.5.1 The effectiveness of our DM approach

First, we evaluate the effectiveness of our DM approach in terms of the number of required processors. Figure 3.2 to 3.4 show the results under global scheduling. The number of processors needed to schedule a task set is computed using Equation (3.2).

Figure 3.2 shows the results with latency constraint $L_0(L_{\min})$. Under such stringent latency constraint, the intervals in which deadlines of actors may vary are limited. Our DM approach is still capable of reducing the number of processors compared to the BA for 8 out of 13 benchmarks. The largest reduction is obtained for the Vocoder benchmark, with a reduction of 66 processors. The DCT, TDE and H.263 benchmarks have only a single data path in the corresponding CSDF graphs. The Beamformer and

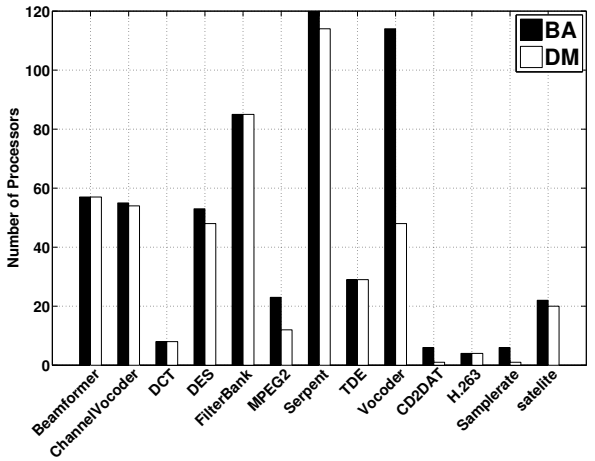

Figure 3.2: Global scheduling with L_0 constraint

Filterbank benchmarks have symmetric graph structures, i.e., multiple paths consist of the same type of actors. Therefore, for all these benchmarks, small intervals in which deadlines of actors may vary restrict the possibility to reduce the total density, consequently the required number of processors is not reduced.

Figure 3.3 presents the results for a relaxed latency constraint for each benchmark. There are 6 benchmarks for which our DM approach reduces the number of processors. In this case, the Beamformer benchmark with symmetric structure, also benefits from the DM approach. That is because the redistribution of deadlines on a path makes it possible to decrease the densities δ_i of some tasks/actors. For DCT, TDE, and H.263, although we can see a reduction in the total density δ_A of the task set, the reduction is smaller and insufficient to decrease the number of processors. The Samplerate and ChannelVocoder benchmarks keep unchanged on the number of processors because the total density is very close to the total utilization which is the lower bound of δ_A. Figure 3.4 shows the results for a very relaxed latency constraint, where only 5 benchmarks get reduction on the number of processors by using our DM approach.

In summary, our DM approach obtains reduction in the number of processor in more than 48% of the conducted experiments.

3.5.2 The time complexity of solving our DM problem

In this section, we evaluate the efficiency of our DM approach in terms of the execution time of CVX [GB14] for solving our DM problem. We set a maximum runtime of 4 hours for the solver, and all results are summarized in Table 3.5 where the time

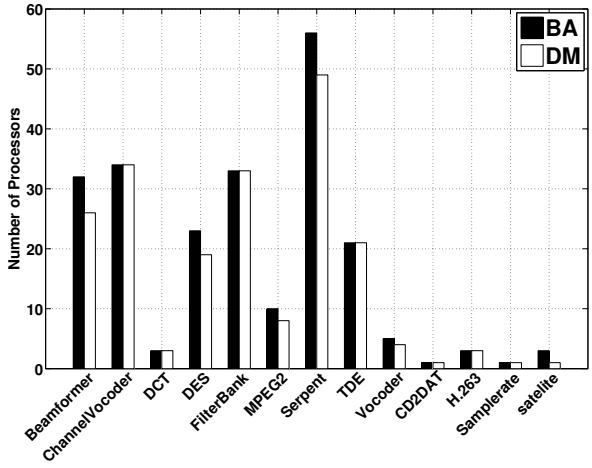

Figure 3.3: Global scheduling with L_1 constraint

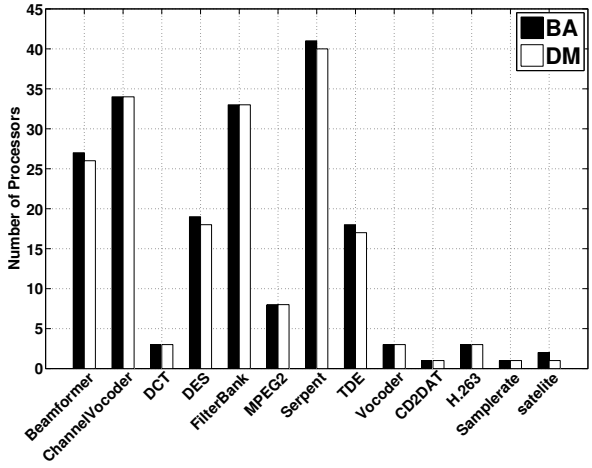

Figure 3.4: Global scheduling with L_2 constraint

Applications	L_0	L_1	L_2
Beamformer	0.05	0.18	0.11
ChannelVocoder	0.07	0.11	0.11
DCT	0.05	0.07	0.06
DES	5.9	14.7	0.23
FilterBank	0.1	0.32	0.17
MPEG2	12.9	0.13	0.07
Serpent	20.59	900.98	4751
TDE	0.13	0.1	0.24
Vocoder	1909	2256	0.31
CD2DAT	0.11	0.21	0.1
H.263	0.22	11.72	0.23
Samplerate	0.14	0.1	0.06
Satelite	0.14	0.08	0.24

Table 3.5: The execution time of our DM approach (in second)

unit is seconds. We can see that the runtime of CVX is not a function of the number of actors and edges in the corresponding application CSDF graph. For example, the FilterBank benchmark with more actors and edges than the DES benchmark just needs 0.32 second to find the optimal solution for L_1, while the DES benchmark needs 14.7 seconds. Additionally, even for the same benchmark with different latency constraints, the runtime for finding the optimal solution is fluctuating significantly. The execution times of our DM approach with the Vocoder benchmark for L_1 and L_2 is 2256 seconds and 0.31 second, respectively. According to Table 3.5 most of the problems can be solved in a second. However, for the two very complex benchmarks, Serpent and Vocoder which have the largest number of constraints, the solver spent a long time to find the optimal solution. A method to speed up solving a very complex problem is to set an initial solution for optimization variables which can be obtained from BA, but unfortunately CVX does not support initialization of the optimization variables.

3.6 Discussion

In this work, we address the resource minimization problem of CSDF-model streaming applications under the global optimal scheduling when considering the *HRT* framework [BS11, BS12]. The proposed DM approach is also applicable to partitioned scheduling. The approach given in [SLCS16] shows how our DM approach can be extended to partitioned scheduling. In summary, since there exists no optimal partitioned scheduling, we cannot directly build a convex objective function for it. There-

fore, in [SLCS16], our DM approach is first used to select optimal deadlines for actors using the same objective function as in Equation (3.9a) and then a partitioned algorithm, e.g., FFD, WFD, etc (explained in Section 2.2.3), is deployed to assign task to processor. If the current number of processors cannot suffice the schedulability of the new input task, the number of processors increments by one. The procedure repeats until all tasks are successfully mapped to the system. *It is worth to notice that when our DM approach is applied in partitioned scheduling, as in [SLCS16], it only derives a suboptimal result.*

Chapter 4

Energy Optimization for Real-Time Streaming Applications

Di Liu, Jelena Spasic, Gang Chen and Todor Stefanov,
"Energy-efficient mapping of real-time streaming applications on cluster heterogeneous MPSoCs,"
The 13th IEEE Symposium on Embedded Systems For Real-time Multimedia (ESTIMedia),
Amsterdam, 2015, pp. 1-10.

With the increasing number of cores/processors fabricated on a chip, the stringent area requirement hinders to implement the core's frequency and voltage regulator hardware by using the fine-grained per-core DVFS due to its high hardware cost [HM07]. In order to balance the energy saving and hardware cost, cluster heterogeneous MPSoCs provide a promising solution. On cluster heterogeneous MPSoCs, all processors on the same cluster operate at the same frequency and voltage level, i.e., processors on the same cluster are managed by a single frequency and voltage regulator. Although the cluster heterogeneous MPSoC systems have shown their energy efficiency in the state-of-the-art chips and commercial products, e.g., Samsung Exynos 5422 [Sam16], Samsung Galaxy S7, Iphone 7 and 7Plus, etc., there has been no sufficient effort by the design community to devise a systematic approach for mapping real-time streaming applications onto a cluster heterogeneous MPSoC. Thus, motivated by this fact, in this chapter, we present our new algorithm to map streaming applications onto a cluster heterogeneous MPSoC, where the streaming applications are modeled as CSDF graphs and scheduled using the HRT scheduling framework in Section 2.3. The proposed novel algorithm is called Frequency Driven Mapping (FDM) and its main novelty is twofold:

1. By using the HRT scheduling framework for CSDF graphs, explained in Section 2.3, we propose an efficient way to determine a suitable processor type for each

actor/task in a CSDF graph, where the energy consumption is minimized and throughput and latency constraints are met;

2. According to an initial mapping derived by the first-fit-decreasing (FFD) heuristic and the properties of cluster MPSoCs, we remap some tasks to unused clusters in order to further reduce the energy consumption by using VFS.

We have performed various experiments on real-life streaming applications. The experimental results show that compared to the existing approaches in [CKR14] and [KYD11], the proposed FDM algorithm can achieve more energy saving.

4.1 Related Work

The energy-efficient design issue has been addressed extensively in the past decade. Voltage/frequency scaling (VFS) and dynamic power management (DPM) are the two major techniques used to achieve energy efficiency. In particular, in real-time systems, a considerable amount of work has been done to deploy VFS to schedule real-time tasks in an energy-efficient way. [CK07] surveys most of the papers dealing with energy efficient scheduling of real-time tasks by using VFS. All papers reported in [CK07] assume per-core VFS platforms. Regardless of the properties of cluster MPSoCs, directly adopting these per-core VFS approaches onto cluster MPSoCs may lead to an energy inefficient design/mapping [KYD11]. In contrast, the algorithm proposed in this chapter is specifically devised for cluster heterogeneous MPSoCs and effectively deals with the mapping problem on the cluster heterogeneous MPSoCs in terms of energy efficiency.

Although the works in [CKR14] and [KYD11] have addressed real-time task mapping on cluster-based MPSoC systems, they are different from our work in several aspects. The differences are summarized in Table 4.1. First, [CKR14] and [KYD11] only consider to meet deadlines for each task, but do not consider the applications' performance constraints, e.g., throughput and latency, which streaming applications usually are subject to. Applying their techniques directly onto the considered streaming applications will not guarantee the performance constraints, i.e., throughput and latency. Second, they consider partitioned scheduling in their work, whereas motivated by the state-of-the-art hardware, i.e., ARM big.LITTLE heterogeneous MPSoCs, we consider a cluster scheduling which provides a good trade-off between global scheduling and partitioned scheduling with respect to schedulability and scalability [BBA10]. Thus, the cluster scheduling can achieve a better system utilization than partitioned scheduling. Finally, the work in [KYD11] only considers homogeneous MPSoCs, whereas we consider heterogeneous MPSoCs which provide better energy efficiency.

	heterogeneous	performance	scheduling
Colin *et al.* [CKR14]	Yes	No	Partitioned
Kong *et al.* [KYD11]	No	No	Partitioned
Our FDM approach	Yes	Yes	Cluster

Table 4.1: The difference from [KYD11] and [CKR14]

Some works have been done to reduce the energy consumption of streaming applications on MPSoCs [SDK13], [XMM07], [WLL+11] and [CHBK13], where some performance metrics are taken into account. The works in [SDK13], [XMM07], [WLL+11] and [CHBK13] use per-core VFS on homogeneous MPSoCs to reduce the energy consumption. In contrast, we consider cluster heterogeneous MPSoCs in our work. Heterogeneous MPSoCs are known to be more energy efficient than homogeneous MPSoCs [KFJ+03]. Since finding a suitable type of processor for each actor/task is really important with respect to energy reduction and guaranteed performance, mapping tasks to heterogeneous MPSoCs is a more challenging job than mapping tasks to homogeneous MPSoCs. Moreover, adopting the per-core VFS approaches without considering the properties of cluster MPSoCs may result in an energy inefficient mapping [KYD11].

4.2 Background

In this section, we elaborate the system model and present the power/energy model considered throughout this chapter.

4.2.1 System Model

We consider a cluster heterogeneous MPSoC which has two types of clusters, namely performance-efficient (PE) clusters and energy-efficient (EE) clusters, which refer to the ARM's **big.LITTLE** architecture [ARM16]. Each cluster has a number of identical processors, PE processor or EE processor. In total, the cluster heterogeneous MPSoC consists of $N_c^{\text{PE}} \times M_p^{\text{PE}}$ PE processors and $M_c^{\text{EE}} \times M_p^{\text{EE}}$ EE processors, where N_c^{PE} and M_p^{PE} represent the total number of PE clusters and the number of processors per PE cluster, respectively. M_c^{EE} and M_p^{EE} are the total number of EE clusters and the number of processors per EE cluster, respectively. Figure 4.1 shows an example of a cluster heterogeneous MPSoC, where there are two PE clusters and two EE clusters, each cluster with four identical processors. The processors within one cluster share some resources, e.g, the last level cache, the memory controller, etc. A cluster in the system can be switched off, thereby consuming no power. In this chapter, we only consider symmetric clusters, i.e., all clusters have the same number of processors, but our

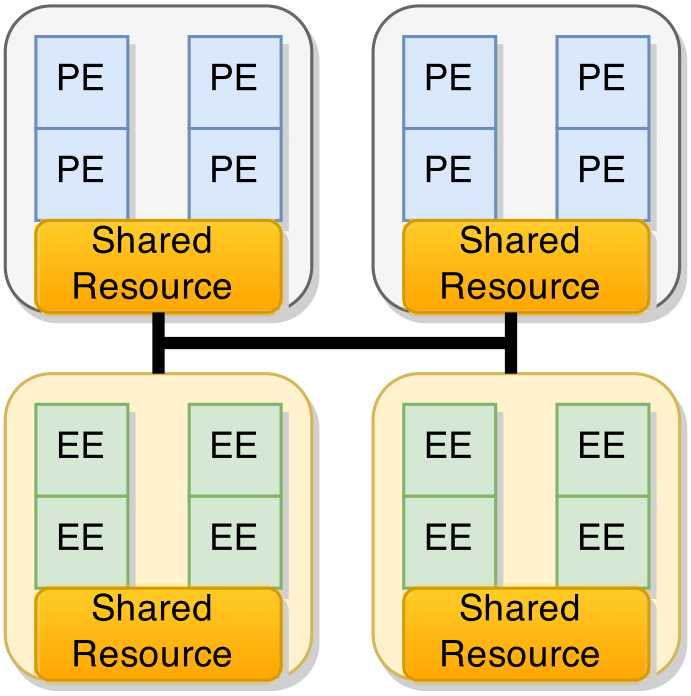

Figure 4.1: An example of a cluster heterogeneous MPSoC

approach can be easily adapted to asymmetric clusters. Since application tasks/actors may run on different processor types, the WCET C_i of a task/actor varies according to the performance of the assigned processor type. Hence, we first extend the task model described in Section 2.3 to support this WCET variation due to the heterogeneity of the systems. We replace the scalar C_i with a vector $\vec{C_i}$ which consists of two WCETs, C_i^{EE} and C_i^{PE}, corresponding to the WCET of a task running on an EE processor and on a PE processor, respectively. C_i^{EE} and C_i^{PE} are the WCETs when EE and PE processors run at their maximum operating frequencies supported by the hardware platform. A real-time task τ_i then is specified by a tuple of $\tau_i = \{S_i, D_i, T_i, C_i^{\text{EE}}, C_i^{\text{PE}}\}$.

As we mentioned in Section 4.1, we adopt cluster scheduling in this chapter. With cluster scheduling, we compute the minimum voltage/frequency level offline and configure the cluster with the computed voltage/frequency level. On each cluster, an optimal global scheduling algorithm, e.g., PFair [AS00] or LLREF [CRJ06], is deployed to schedule actors. A periodic taskset is schedulable on a homogeneous multiprocessor system by an optimal global scheduling algorithm, if $U = \sum_{\forall \tau_i \in \Gamma} C_i/T_i \leq M$ [BCPV93a], where U is the total utilization of the taskset and M is the number of processors. Since in our work we consider an optimal global scheduling algorithm for

each cluster, tasks mapped onto a PE cluster are schedulable if $U_k^{PE} \leq M_p^{\text{PE}}$, while tasks mapped onto an EE cluster are schedulable if $U_j^{EE} \leq M_p^{\text{EE}}$. U_k^{PE} and U_j^{EE} are the utilization of PE cluster k and EE cluster j, respectively. They are computed by the following equations:

$$U_j^{\text{EE}} = \sum_{\forall \tau_i \in A_j^{\text{EE}}} \frac{C_i^{\text{EE}}}{\breve{T}_i}, \qquad U_k^{\text{PE}} = \sum_{\forall \tau_i \in A_k^{\text{PE}}} \frac{C_i^{\text{PE}}}{\breve{T}_i} \tag{4.1}$$

where A_j^{EE} and A_k^{PE} represent the set of actors assigned to EE cluster j and PE cluster k, respectively.

4.2.2 Energy Model

In this chapter, we use real measurements from the ODROID XU-3 [ODR16] board to build our power and energy models. The ODROID XU-3 has an Exynos 5422 chip [Sam16], where there are two clusters on the chip, one quad core Cortex A15 (big) and one quad core Cortex A7 (LITTLE). The power consumption of a cluster consists of two parts, 'processor' and 'uncore' [GBK+12]. The 'processor' power consumption is power dissipated by the processors, while the 'uncore' power consumption is the power consumption from some components not pertaining to a processor, e.g., a shared cache, an integrated memory controller, etc. The 'uncore' power consumption of the ODROID XU-3 is shown in Table 4.2, where the 'uncore' power consumption for the PE (big) cluster and EE (LITTLE) cluster, at different operating frequencies, are given. We can see that for the PE (big) cluster the 'uncore' power consumption $P_s^{\text{PE}}(f)$ scales along with the cluster operating frequency. We find that the 'uncore' power consumption $P_s^{\text{PE}}(f)$ contributes approximately 20% to the total power consumption of the big cluster. For the EE (LITTLE) cluster, with the on-chip power sensor, we can not see the variation of the 'uncore' power consumption $P_s^{\text{EE}}(f)$ at different frequency levels. Hence, in Table 4.2, the 'uncore' power consumption $P_s^{\text{EE}}(f)$ is the same for each frequency level. Note that since one core on the LITTLE cluster has to be active for the operating system, we are not able to measure the pure 'uncore' power consumption $P_s^{\text{EE}}(f)$ for the LITTLE cluster. The values of $P_s^{\text{EE}}(f)$ given in Table 4.2 include some power consumption from the active core running the OS.

Although the 'uncore' power consumption may be related to the frequency as shown in Table 4.2, it is different from the dynamic power consumption which also relates to the frequency. Dynamic power is only consumed when there is a workload on the processors, whereas the 'uncore' power consumption always exists as long as the cluster is on. Based on the above discussion, we use the following power model for each cluster,

$$P(f) = \alpha f^b + M_p \beta + P_s(f) \tag{4.2}$$

f (GHz)	2	1.8	1.6	1.4	1.2	1	0.8
$P_s^{\text{PE}}(f)$ (W)	0.8	0.528	0.39	0.309	0.244	0.182	0.134
f (GHz)	1.4	1.2	1	0.8	0.6	0.4	0.2
$P_s^{\text{EE}}(f)$ (W)	0.04	0.04	0.04	0.04	0.04	0.04	0.04

Table 4.2: The 'uncore' power consumption

where the first term is the dynamic power consumption, β is the static power consumption of one processor and M_p is the number of processors on the cluster. $P_s(f)$ is the 'uncore' power consumption and f is the frequency level. In this chapter, we use power parameters from ODROID XU-3 as our reference, so parameters α, b, and β are estimated by using curve fitting with real power measurements from the ODROID XU-3 board. The estimated parameters for each processor type are shown in Table 4.3.

processor type	α (W/MHz^b)	b	β (W)
PE (big)	3.03×10^{-9}	2.621	0.155
EE (LITTLE)	2.62×10^{-9}	2.12	0.0278

Table 4.3: The estimated parameters

To validate our power model, described above, we measure the power consumption from the board and compare it with the estimations obtained from our model. We keep one core on and run a computation-intensive job on the core. Then, we measure the power consumption at different frequency levels for each cluster through the on-chip power sensors. Figure 4.2 plots two curves for the PE (big) cluster, one for the measured power consumption and another for the estimated power consumption, while Figure 4.3 plots two curves for the EE (LITTLE) cluster, one for the measured power consumption and another for the estimated power consumption. In the two figures, the y-axis shows the power consumption, while the x-axis shows the different operating frequency levels. From the figures, we can see that the estimated curves are close to the measured curves, so our power model is sufficiently accurate.

To compute the total system energy consumption, we need to introduce the concept of a *hyper-period* (hp),

$$hp = \text{lcm}(\breve{T}_1, \breve{T}_2, , \ldots, \breve{T}_i) \tag{4.3}$$

where the lcm is the *least common multiple* and $\breve{T}_1, \breve{T}_2, \ldots, \breve{T}_i$ are the minimum periods of application actors/tasks computed using Equation (2.8). For periodic tasks,

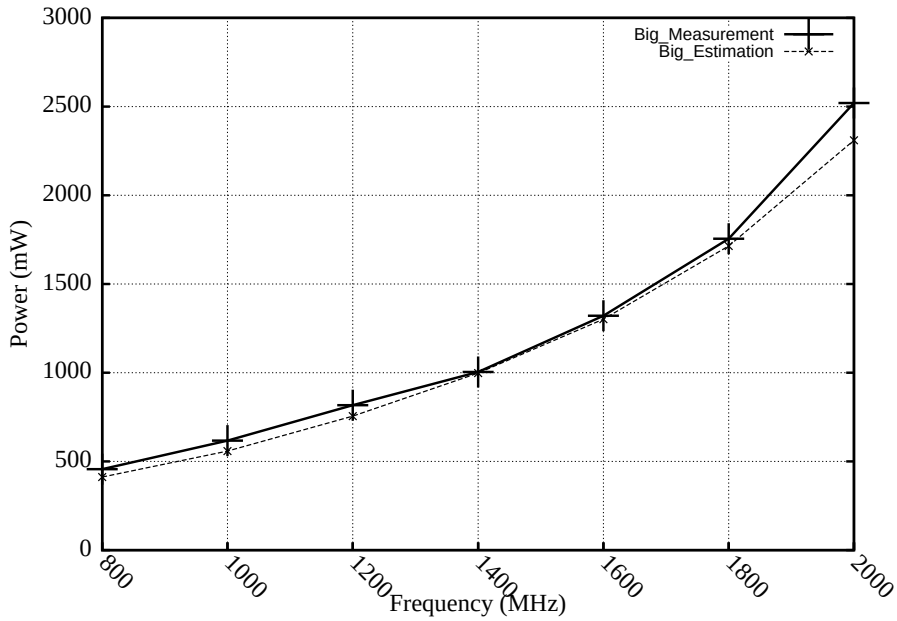

Figure 4.2: Power model validation of PE (big) cluster

every *hyper-period* has the same workload, i.e, all tasks will execute for a certain number of times. Hence, with the definition of the *hyper-period*, we can build the energy model of an MPSoC within one *hyper-period* as follows:

$$E = E_s + E_d \tag{4.4}$$

where E_s is the total static energy consumption and the total 'uncore' energy consumption which is computed as:

$$E_s = hp\Big(\sum_{j=1}^{M_{\text{ac}}^{\text{EE}}} {P_s}^{\text{EE}}(f_j) + \sum_{k=1}^{M_{\text{ac}}^{\text{PE}}} P_s^{\text{PE}}(f_k) + M_{\text{ac}}^{\text{EE}} M_p^{\text{EE}} \beta^{\text{EE}} + M_{\text{ac}}^{\text{PE}} M_p^{\text{PE}} \beta^{\text{PE}}\Big) \tag{4.5}$$

$M_{\text{ac}}^{\text{EE}}$ is the number of active EE clusters and $M_{\text{ac}}^{\text{PE}}$ denotes the number of active PE clusters. M_p^{PE} and M_p^{EE} denote the number of processor per PE cluster and EE cluster, respectively. f_j and f_k are the operating frequency levels for the corresponding EE cluster and PE cluster, respectively. β^{EE} and β^{PE} are the power parameters shown in the last column of Table 4.3.

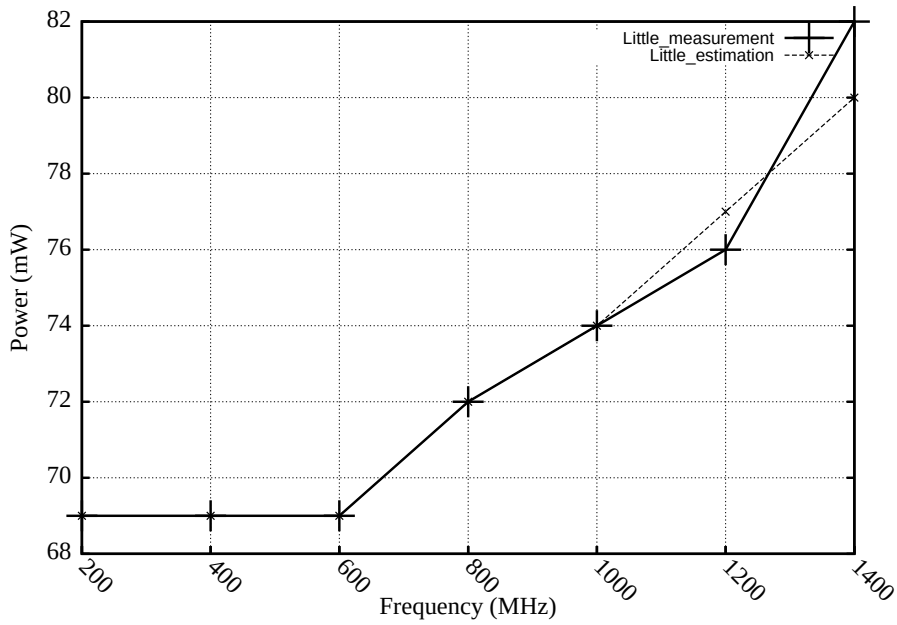

Figure 4.3: Power model validation of EE (LITTLE) cluster

The total dynamic energy consumption E_d in Equation (4.4) is computed as:

$$\begin{aligned} E_d &= hp\Big(\sum_{j=1}^{M_{\text{ac}}^{\text{EE}}} \sum_{\forall \tau_i \in A_j^{\text{EE}}} \frac{C_i^{\text{EE}}}{T_i} \alpha^{\text{EE}} f_j^{b_{\text{EE}}} \frac{f_j^{\max}}{f_j} + \sum_{k=1}^{M_{\text{ac}}^{\text{PE}}} \sum_{\forall \tau_i \in A_k^{\text{PE}}} \frac{C_i^{\text{PE}}}{T_i} \alpha^{\text{PE}} f_k^{b_{\text{PE}}} \frac{f_k^{\max}}{f_k}\Big) \\ &= hp \sum_{j=1}^{M_{\text{ac}}^{\text{EE}}} U_j^{\text{EE}} \alpha^{\text{EE}} f_j^{(b_{\text{EE}}-1)} f_j^{\max} + hp \sum_{k=1}^{M_{\text{ac}}^{\text{PE}}} U_k^{\text{PE}} \alpha^{\text{PE}} f_k^{(b_{\text{PE}}-1)} f_k^{\max} \end{aligned} \tag{4.6}$$

where f_j and f_k are the operating frequencies of the corresponding EE cluster and PE cluster, respectively. Correspondingly, $f_j^{\max}$ and $f_k^{\max}$ are the maximum operating frequencies of the EE and PE cluster, respectively. α^{EE}, b_{EE}, α^{PE} and b_{PE} are the estimated power parameters for an EE cluster and a PE cluster, shown in Table 4.3.

4.3 Proposed Mapping Algorithm

In this section, we present our mapping algorithm, called Frequency Driven Mapping (FDM), which is able to energy-efficiently map real-time streaming applications, modeled as CSDF graphs, to cluster heterogeneous MPSoCs while guaranteeing the

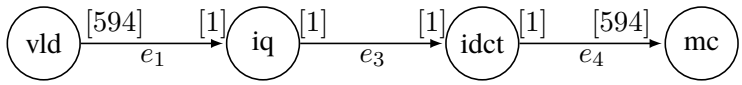

Figure 4.4: H.263 Decoder

throughput and latency constraints. The complete FDM algorithm is given in Algorithm 5 in Section 4.3.4. Before we explain FDM in details, we would like to introduce the three foundations of FDM which are described in Section 4.3.1, 4.3.2, 4.3.36.

4.3.1 Processor Type Assignment

In a heterogeneous MPSoC, choosing the type of processor for each task in an application is crucial. To support this statement, we show an example with a real-life streaming application, H.263 decoder, which is modeled as an SDF graph which is a subset of CSDF. Hence, the HRT scheduling explained in Section 2.3 is applicable to the SDF. The SDF-modeled H.263 decoder consists of 4 tasks/actors and 3 edges, as shown in Figure 4.4. The parameters of each actor in the H.263 decoder are shown in Table 4.4, where C_i^{PE} and C_i^{EE} denote the WCETs in clock cycles (cc) of the actor on the PE cluster and EE cluster, respectively. q_i is the repetition value which is used to compute the workload explained in Definition 2.3.1.

	$C_i^{\text{PE}}(cc)$	$C_i^{\text{EE}}(cc)$	q_i
vld	26018	52036	1
iq	559	1118	594
idct	500	1000	594
mc	10958	21916	1

Table 4.4: The parameters of H.263

In Table 4.5, different processor type assignments for all actors are presented. Column 1 shows the number identifying processor type assignments, and column 2 to 5 show which type of processor each actor is assigned to where we highlight the EE processor type. The last two columns show the latency and the throughput of these processor type assignments, which are computed by using Equation (2.11) and Equation (2.12) and the parameters in Table 4.4. In these two columns, we highlight the values which satisfy the latency and throughput constraints. Intuitively, we want to have more actors running on EE processors as long as the latency and throughput constraints are met. According to the figures in Table 4.5, we can see that an inappropriate processor type assignment significantly degrades the system performance and violates

	Processor Type Assignment				L	$\mathcal{R}$(token/
	vld	*iq*	*idct*	*mc*	$(cycles)$	$cycles)$
1	**EE**	PE	PE	PE	**996697**	**1/332046**
2	PE	**EE**	PE	PE	1993394	1/664092
3	PE	PE	**EE**	PE	1783000	1/594000
4	PE	PE	PE	**EE**	**996697**	**1/332046**
5	**EE**	PE	PE	**EE**	**996697**	**1/332046**

Table 4.5: Different processor type assignments for the H.263 decoder

the constraints. Looking at processor type assignments 2 and 3, assigning either task *iq* or *idct* to the EE type of processor leads to a violation of the performance constraints. On the other hands, processor type assignment 5 assigns both tasks *vld* and *mc* to the EE type of processor while the performance constraints are met. Thus, determining a good processor type assignment is essential for heterogeneous MPSoCs.

From the example given above, in order to efficiently assign actors to processor types, it is important to identify those tasks which will violate the performance constraints if assigning them to the EE type of processor. By considering the characteristics of the HRT scheduling of CSDF, we propose an efficient way to split tasks into two categories, bottleneck and non-bottleneck tasks. The bottleneck actors/tasks should be assigned to PE processors in order to guarantee the performance, while the non-bottleneck actors/tasks can be assigned to EE processors for the purpose of energy saving. We introduce the following proposition:

Proposition 1. *For a CSDF graph scheduled using hard-real-time scheduling, increasing WCET C_i of task τ_i will not increase the latency and reduce the throughput, if the maximum workload $\hat{W}$ remains the same.*

Proof. By looking at Equation (2.11), we can see that the latency is only determined by the start times and periods of the input and output actors. On the one hand, from Equation (2.8), it is not difficult to see that $\hat{W}$ is the variable part to compute period $\check{T}_i$, because q_i and $lcm(\overrightarrow{q})$ are both constants. Hence, as long as the maximum workload $\hat{W}$ does not increase, increasing other actors' WCETs will not change any actor's period. As a result, the throughput will not be reduced. On the other hand, start time S_i depends on the data-dependency and the deadlines of precedent actors. The data-dependency will not change in any case, while $D_i = \check{T}_i$ and period $\check{T}_i$ does not change. Hence, S_i remains the same as well. As a result, the latency does not increase. □

It follows from Proposition 1 that some actors in the graph can execute slowly, while not degrading the application performance. Thus, Proposition 1 can help us to

Algorithm 2: Processor Type Assignment

Input: $G = (A, \mathcal{E})$
Output: A^{EE} and A^{PE}
1 $A \leftarrow$ Sort $\forall \tau_i \in A$ in increasing order of W_i^{EE} of τ_i
2 $b \leftarrow$ Binary search to find the position in A with the biggest index , where actor τ_i can meet $W_i^{\text{EE}} \leq \hat{W}^{\text{PE}}$.
3 $A^{\text{EE}} \leftarrow A[0 : b]$
4 $A^{\text{PE}} \leftarrow A - A^{\text{EE}}$
5 **return** A^{EE} and A^{PE}

classify the actors into the two categories mentioned above. If an actor is assumed to be executed on an EE processor (longer WCET) and its new workload W_i does not change the maximum workload $\hat{W}$, then it is a non-bottleneck actor and can be assigned to an EE cluster without degrading the application performance. Otherwise, the actor should be assigned to a PE cluster in order to guarantee the performance. We look back at the example of the H.263 decoder. From Table 4.5, since executing *vld* and *mc* on the EE type of processor does not lead to an increase of $\hat{W}$, this assignment does not violate the performance constraints. Therefore, we can use $\hat{W}^{\text{PE}}$ as a threshold to determine which type of processor an actor should be run on, where $\hat{W}^{\text{PE}}$ is the **maximum actor workload** assuming that all actors run on PE processors. Algorithm 2 presents a pseudo-code showing how to classify the actors, where A^{EE} and A^{PE} denote the set of actors assigned to EE type and PE type of processors, respectively. To reduce the complexity of the processor type assignment, first we sort the actors in order of increasing workload assuming all of them are assigned to EE processors - see Line 1 in Algorithm 2. Then, with the sorted actors, we use $\hat{W}^{\text{PE}}$ as a threshold and deploy a binary search algorithm to find the pivotal point by which we can split the sorted actors into two sets, one for the EE type of processor and another for the PE type of processor. Since it is impossible to guarantee that the binary search can always find one actor whose W_i^{EE} is equal to $\hat{W}^{\text{PE}}$, we pick up the one with the biggest index as the pivotal point, where the condition $W_i^{\text{EE}} \leq \hat{W}^{\text{PE}}$ is met. Since the sorting algorithm has a complexity of $O(|A| \log |A|)$ and the complexity of the binary search is $O(\log |A|)$, the complexity of Algorithm 2 is $O(|A| \log |A|)$.

The processor type assignment can: (1) assign actors of an application CSDF graph to two different types of processors and (2) allow to initially decide whether the system has enough resources to schedule this application. Suppose that U^{EE} and U^{PE} are the total utilization of A^{EE} and A^{PE} returned by Algorithm 2, respectively. If $U^{\text{EE}} > M_c^{\text{EE}} \times M_p^{\text{EE}}$, the tasks from A^{EE} are not schedulable on EE clusters. If tasks on the EE clusters are not schedulable, we can move some of them to PE clusters such

	PE Clusters	Actor Mapping				Energy Consumption
		vld	*iq*	*idct*	*mc*	(μJ)
WFD	2	PE1	PE2	PE1	PE1	1378
FFD	1	PE1	PE1	PE1	PE1	1226

Table 4.6: Different mappings for H.263 decoder

that the tasks can run on the system. Based on Proposition 1, it is trivial to observe that reassigning some of the tasks in set A^{EE} to set A^{PE}, i.e., assigning these tasks to the PE type of cluster, is still able to guarantee the performance constraints. However, if tasks on the PE clusters are not schedulable, i.e., $U^{\text{PE}} > M_c^{\text{PE}} \times M_p^{\text{PE}}$, that means that with the throughput and latency constraints the application is not schedulable on the system.

4.3.2 Task mapping

When the processor type assignment is determined as described in Section 4.3.1, tasks need to be mapped onto clusters. The task mapping is analogous to a bin-packing problem which is known to be an NP-hard problem [GJ79]. Several well-known heuristic algorithms for the bin-packing problem, e.g., first-fit, best-fit, etc, have been proposed see Section 2.2.3. In terms of energy efficiency, the worst-fit-decreasing (WFD) algorithm is evaluated as the best mapping heuristic [AY03] for the partitioned scheduling. [KYD11] also uses an WFD-like approach. However, by using the following example, we will show that WFD does not work very well in the context of a cluster MPSoC with cluster scheduling.

Here, we use a cluster homogeneous MPSoC to illustrate this problem, where the homogeneous MPSoC has two PE clusters, each with four identical PE processors. Table 4.6 shows two mappings for the H.263 decoder in Figure 4.4 by using two different mapping algorithms, worst-fit-decreasing (WFD) and first-fit-decreasing (FFD). The mapping derived by WFD consumes more energy than the mapping obtained by FFD. The reason is that WFD tries to distribute the heavy tasks to different clusters, where these heavy tasks have large utilization and need a high operating frequency in order to meet their deadlines. In the context of a cluster MPSoC, these heavy tasks constrain the minimum operating frequency of the cluster. On the contrary, FFD always strives to find the first available cluster to map, where this strategy is more likely to map the heavy tasks with the same or close utilization to the same cluster. Then, the system can efficientlly utilize cluster VFS to reduce the energy consumption. Hence, in our approach we use FFD to map tasks to clusters in order to obtain an initial mapping. Given this initial mapping from FFD, we can compute the minimum operating frequency of a cluster. However, the frequency of a cluster is not only determined by

the task with the largest utilization, but also the total utilization of the tasks has to be taken into account. The following example shows the effect of the total utilization.

Example 4.3.1. Consider a cluster with two processors and three tasks with utilization $\{0.5, 0.5, 0.5\}$. The three tasks can be mapped to the cluster because the total utilization of the tasks is $0.5 + 0.5 + 0.5 < 2$. The frequency of this cluster however can not be set according to the task's maximum utilization which is 0.5, because the total utilization is 1.5 and the utilization bound (the number of processor) is 2. If the frequency is scaled with 0.5, then the total utilization of this task set becomes $\frac{1.5}{0.5} > 2$ which means the task set is not schedulable on the cluster.

Considering the example above, the frequency of a cluster should be computed as follows:

$$f_j = \max\left(\max_{\forall \tau_i \in A_j}\{u_i\}, \frac{U_j}{M_p}\right) \times f_{\max} \tag{4.7}$$

where u_i is the utilization of task $\tau_i \in A_j$ and A_j is the set of tasks mapped to cluster j. $f_{\max}$ is the maximum frequency of the type of processor used in the cluster. U_j is the total utilization of A_j and M_p is the number of processors in the cluster. Usually, a cluster only supports a set of finite discrete frequency levels. Hence, we select the minimum frequency from the frequency set which is greater than or equal to frequency f_j in Equation (4.7).

With Equation (4.7), we classify the clusters into two categories, namely U-cluster and T-cluster, which later will be used in our remapping phase described in Section 4.3.3.

Definition 4.3.1. An U-cluster is a cluster where $\max_{\forall \tau_i \in A_j}\{u_i\} < \frac{U_j}{M_p}$.

U-cluster means that the operating frequency of the cluster is determined by the total utilization.

Definition 4.3.2. A T-cluster is a cluster where $\max_{\forall \tau_i \in A_j}\{u_i\} \geq \frac{U_j}{M_p}$.

T-cluster means that the operating frequency of the cluster is determined by the task which has the largest utilization. Hence, we call such task a **constrained task**.

Definition 4.3.3. In a T-cluster, a **constrained task** is the task which has the largest utilization.

4.3.3 Remapping

The FFD algorithm mentioned in Section 4.3.2 enables to quickly map tasks to clusters. On a given MPSoC platform, FFD might just use a few clusters to run the application tasks and leave the rest of the available clusters unused. This would lead to

a few used clusters with high utilization whose frequency can not be scaled down by using VFS. Hence, based on the FFD mapping, we propose a remapping approach to explore the possibility to energy-efficiently utilize the unused clusters on the system such that we can balance or offload the workload of some clusters to the unused clusters in order to further scale down the clusters' operating frequencies to reduce the total system energy consumption.

Our remapping approach is based on analysis for the U-cluster and T-cluster categories introduced and defined in Section 4.3.2. Below, we discuss how to remap tasks for both categories of clusters in order to reduce the energy consumption.

U-cluster

According to Definition 4.3.1, in an U-cluster, the frequency of the cluster is determined by the total utilization. Hence, we strive to remap some tasks to an unused cluster to reduce the total utilization, which in turn allows to scale down the frequency further. In order to minimize the energy consumption, we need to find how many tasks should be remapped and what the optimal frequencies are for the initial cluster and the new cluster to be used.

Consider that we have an initial U-cluster onto which a task set with utilization U is mapped. We split the task set into two subsets, one with utilization U_1 and another with U_2. We remap the task set with U_2 to an unused cluster, and keep the task set with U_1 on the initial cluster. Then the energy consumption of the new mapping can be computed as follows:

$$E = hp\big(U_1\alpha f_1^{(b-1)} f_1^{\max} + M_p\beta + P_s(f_1) + U_2\alpha f_2^{(b-1)} f_2^{\max} + M_p\beta + P_s(f_2)\big) \quad (4.8)$$

where f_1 is the operating frequency of the initial cluster, and f_2 is the operating frequency of the new cluster. In Equation (4.8), there are four variables, U_1, U_2, f_1, and f_2. Since frequencies f_1 and f_2 depend on U_1 and U_2, respectively, and U_1 is related to U_2, these interrelationship between them makes it difficult to find optimal values for all variables in order to minimize Equation (4.8). Hence, with the consideration of simplifying the procedure, we use a load balancing approach to split tasks on an U-cluster into two tasksets. The split tasksets have close total utilizations. Algorithm 3 presents the pseudo-code of splitting the tasks for an U-cluster. We first sort tasks in order of decreasing utilization. Then, we assign the tasks one by one to the taskset Γ_2. As soon as the utilization U_2 of Γ_2 is greater than or equal to the utilization U_1 of Γ_1, the algorithm terminates and returns Γ_1 and Γ_2. Due to the sorting algorithm used in Line 1, the complexity of Algorithm 3 is $O(|\Gamma| \log(|\Gamma|))$.

The remapping will switch on a new cluster, so the remapping should provide enough energy reduction to compensate the static and 'uncore' power consumption of the new cluster. In order to test whether it is worthwhile remapping tasks to an

Algorithm 3: Split tasks for U cluster

Input: A task set Γ
Output: two tasksets Γ_1 and Γ_2

```
Sort tasks in Γ in order of decreasing utilization;
Γ1 ← Γ,   Γ2 ← ∅;
for i = 1 to |Γ| do
    if U2 ≤ U1 then
        Γ1 ← Γ1 − τi;
        Γ2 ← Γ2 + τi;
    else
        Break;
return Γ1 and Γ2;
```

unused cluster, we provide the following proposition to validate the efficiency of the remapping.

Proposition 2. *Given a taskset Γ and its subsets Γ_1 and Γ_2, their utilizations are U, U_1, and U_2, respectively, where $U_1+U_2 = U$, and taskset Γ is assigned to one cluster. The cluster is an U-cluster. Then, moving taskset Γ_2 to an unused cluster can reduce the energy consumption, if the following condition is met:*

$$\begin{aligned}&\left(U_1\alpha(f^{(b-1)} - f_1^{(b-1)}) + U_2\alpha(f^{(b-1)} - f_2^{(b-1)})\right)f^{\max} \\ >&P_s(f_1) + P_s(f_2) + M_p\beta - P_s(f)\end{aligned} \tag{4.9}$$

where f is the operating frequency of the initial cluster before remapping, and f_1 and f_2 are the operating frequencies of the initial cluster and the new cluster after remapping, respectively. $f^{\max}$ is the maximum operating frequency of the cluster.[1]

Proof. Before the remapping, the energy consumption of the initial cluster is as follows:

$$E = hp\big(U\alpha f^{(b-1)} f^{\max} + M_p\beta + P_s(f)\big) \tag{4.10}$$

After the remapping, the energy consumption of the two clusters is:

$$E_n = hp\big(U_1\alpha f_1^{(b-1)} f^{\max} + M_p\beta + P_s(f_1) + U_2\alpha f_2^{(b-1)} f^{\max} + M_p\beta + P_s(f_2)\big) \tag{4.11}$$

[1] Since we only do remapping on the same type of cluster, the maximum operating frequency is the same.

To guarantee that the remapping leads to less energy consumption, we need the following inequality satisfied:

$$E > E_n$$

Since we consider symmetric clusters, by substituting Equation (4.10) and (4.11) in the inequality and eliminating the same terms on both sides, we obtain the following condition:

$$\begin{aligned} &\left(U_1\alpha(f^{(b-1)} - f_1^{(b-1)}) + U_2\alpha(f^{(b-1)} - f_2^{(b-1)})\right) f^{\max} \\ >&P_s(f_1) + P_s(f_2) + M_p\beta - P_s(f) \end{aligned} \tag{4.12}$$

□

T-cluster

According to Definition 4.3.2, in a T-cluster, the frequency of the cluster is determined by the **constrained task** (Definition 4.3.3). However, within one cluster, the FFD discussed in Section 4.3.2 might map some other tasks which have lower utilization and could operate at a lower frequency. In this case, remapping these tasks to an unused cluster operated at a lower frequency may result in an overall reduced energy consumption.

Example 4.3.2. Given two clusters with two processors each and a task set with three tasks where the utilizations are $\{0.9, 0.3, 0.3\}$, the FFD maps all tasks to one cluster, and the cluster is a T-cluster. The frequency is determined by the task with utilization 0.9. However, if we map the two tasks with utilization 0.3 to the unused cluster, the frequency of the initial cluster is not changed, but the new cluster operates at a lower frequency which can significantly reduce the overall energy consumption.

Example 4.3.2 illustrates how we can remap tasks of a T-cluster. We find the actors which can run at a frequency lower than the current cluster frequency and remap them to an unused cluster. Algorithm 4 presents the pseudo-code of splitting tasks of a T-cluster. The complexity of Algorithm 4 is $O(|\Gamma| \log |\Gamma|)$ due to the sorting algorithm used in Line 1. The remapping will need more static power consumption and 'uncore' power consumption due to the new cluster switched on. Thus, the efficiency of the remapping should be verified. The following proposition presents an efficient way to check this.

Proposition 3. *Given a taskset Γ and its subsets Γ_1 and Γ_2, their utilizations are U, U_1, and U_2 respectively, where $U = U_1 + U_2$, and taskset Γ is assigned to one cluster. The cluster is a T-cluster and the constrained actor is in subset Γ_2. Then, moving*

Algorithm 4: Split tasks for T-cluster

Input: A task set Γ
Output: two tasksets $\Gamma 1$ and Γ_2

```
Sort tasks in Γ in order of decreasing utilization;
Γ_1 ← ∅,   Γ_2 ← ∅;
for i = 1 to |Γ| do
    if τ_i can run at a lower frequency then
        for j = i to |Γ| do
            Γ_1 ← Γ_1 + τ_j;
        Γ_2 ← Γ − Γ_1;
        Break;
return Γ_1 and Γ_2;
```

taskset Γ_1 to an unused cluster can reduce the energy consumption, if the following condition is met:

$$U_1 \cdot \alpha \cdot f^{(b-1)} f^{\max} > U_1 \cdot \alpha \cdot f_1^{(b-1)} f^{\max} + M_p\beta + P_s(f_1) \tag{4.13}$$

where f and f_1 are the operating frequencies of the initial and new cluster, respectively. $f^{\max}$ is the maximum operating frequency of the cluster. [2]

Proof. Assume that taskset Γ is assigned to one cluster and its operating frequency is f. Before the remapping, the energy consumption of the initial cluster can be computed as follows:

$$E = hp\big(U \cdot \alpha \cdot f^{(b-1)} f^{\max} + M_p\beta + P_s(f)\big) \tag{4.14}$$

If taskset Γ_1 which is a subset of Γ is remapped to an unused cluster, the operating frequency of the new cluster is f_1. Since the constrained task is in taskset Γ_2 and taskset Γ_2 remains on the initial cluster, the frequency of the initial cluster does not change. After the remapping, the energy consumption of the two clusters is the following:

$$E_n = hp\big(U_2 \cdot \alpha \cdot f^{(b-1)} f^{\max} + M_p\beta + P_s(f) + U_1 \cdot \alpha \cdot f_1^{(b-1)} f^{\max} + M_p\beta + P_s(f_1)\big) \tag{4.15}$$

The assignment with two clusters is more energy-efficient, if $E > E_n$. Since $U = U_1 + U_2$, we replace U_2 with $U - U_1$ in Equation (4.15). By substituting Equation (4.14) and (4.15) and eliminating the same terms on both sides of inequality $E > E_n$, we obtain:

$$U_1 \cdot \alpha \cdot f^{(b-1)} f^{\max} > U_1 \cdot \alpha \cdot f_1^{(b-1)} f^{\max} + M_p\beta + P_s(f_1) \tag{4.16}$$

[2] Same type of clusters has the same maximum operating frequency.

□

4.3.4 The FDM Algorithm

In this section, we present our overall mapping algorithm, called Frequency Driven Mapping (FDM). The inputs to FDM are a CSDF graph $G = (A, \mathcal{E})$ and a cluster heterogeneous MPSoC, and the outputs are the task mapping to clusters and the minimum operating frequency for each cluster which is active. Algorithm 5 shows the pseudo-code of FDM. In Line 1, FDM applies Algorithm 2 explained in Section 4.3.1 to split tasks into two sets A^{PE} and A^{EE} which denote set of the tasks assigned to PE type and EE type of processors, respectively. In Algorithm 2, the processor type assignment completes the assignment procedure with the guarantees of meeting the performance constraints. Hence, if the task sets A^{EE} and A^{PE} derived by Algorithm 2 are schedulable on the given MPSoC, the throughput and latency constraints are met for the application. From Line 2 to 5, we check whether the input MPSoC has enough resources to schedule the real-time streaming application. If there is no enough EE type of processors, we select some tasks from set A^{EE} and assign them to set A^{PE} such that we have enough EE processors to schedule the tasks in set A^{EE}. The tasks are selected in order of decreasing utilization, and the selection is terminated as soon as the tasks in set A^{EE} are schedulable on the EE processors. However, if there is no enough PE type of processors, this means the application is not schedulable on the input MPSoC. The algorithm terminates and signals failure at Line 5. After this schedulability check, we use the FFD heuristic discussed in Section 4.3.2 on PE clusters and EE clusters to map tasks to clusters in Line 6. After this phase, we obtain the initial mapping and the corresponding set of active clusters, i.e., Φ_{ac} shown in Line 6. An element in set Φ_{ac} is a cluster which includes the tasks mapped to the cluster and the operating frequency of the cluster computed by Equation (4.7).

With the obtained initial mapping, a remapping procedure starts from Line 7. In this procedure, we go through every cluster in set Φ_{ac} to check what category the cluster falls into, U-cluster or T-cluster. From Line 8 to 14, we do remapping for an U-cluster. At Line 8 and 9, if a cluster is an U-cluster of type EE or PE and there is an unused cluster of the same type available (PE or EE), we split the tasks into two sets by using Algorithm 3 and we use Proposition 2 to validate the remapping in Line 10. If the remapping leads to energy reduction, we complete the remapping. Otherwise, the mapping remains unchanged. From Line 15 to 21, we do remapping for a T-cluster. For a T-cluster, we use Algorithm 4 to split tasks in Line 16 and we use Proposition 3 to validate the remapping in Line 17. Note that after we do a remapping the new cluster ϕ_{unused} is added to the active cluster set Φ_{ac} shown in Line 12 and 19. Then later the new cluster also will undertake the remapping procedure as long as there is an unused cluster of the same type and it can meet the remapping conditions.

Finally, FDM updates the operating frequencies of each cluster in set Φ_{ac} in Line 22 by using Equation (4.7). At Line 23, FDM outputs the final mapping and the operating frequency of each active cluster. Since the complexity of Algorithm 3 and 4 are both $O(|A|\log(|A|))$, in the worst case the complexity of FDM is $O(N \times |A|\log(|A|))$, where N is the total number of clusters in the input MPSoC and $|A|$ is the total number of actors in the input CSDF graph.

4.4 Evaluation

In this section, we present three experiments to demonstrate the efficiency of the proposed FDM algorithm compared to the existing approaches proposed in [CKR14] and [KYD11]. We apply our FDM and the mapping approaches from [CKR14] and [KYD11] on cluster heterogeneous and homogeneous MPSoCs. We choose [CKR14] and [KYD11] to compare with, because the mapping approaches in [CKR14] and [KYD11] are specifically devised for cluster MPSoCs as considered in our work. Therefore, their work is the most related and relevant to our approach.

We select 11 real-life streaming applications from the StreamIt [TA10] benchmark suite and the MP3 decoder [SGB06], where all streaming applications are modeled as CSDF graphs. We use the same parameters, i.e., WCETs of application tasks, as specified in [BS11]. An overview of all streaming applications is given in Table 4.7. $|A|$ denotes the number of tasks/actors in a CSDF graph, while $|\mathcal{E}|$ denotes the number of edges. L is the minimum achievable latency and $\mathcal{R}$ is the maximum achievable throughput which are computed by using Equation (2.11) and Equation (2.12) in Section 2.3, when the applications are scheduled by the HRT scheduling. In our experiments, for each application, we set as constraints the corresponding minimum achievable latency and the maximum achievable throughput (L and $\mathcal{R}$ given in Table 4.7) when we map the applications to the target platforms.

As target platforms, we consider three heterogeneous MPSoCs with different number of clusters and cluster granularities. We use 'MPSoC_x_pe_ee' to denote a cluster heterogeneous MPSoC, where 'x' denotes the number of processors per cluster, 'pe' and 'ee' denote the number of PE clusters and EE clusters, respectively. The three considered MPSoCs are described in Table 4.8. Column 'granularity' shows the number of processors per cluster, while column 'PE clusters' and 'EE clusters' show the number of PE clusters and EE clusters in the MPSoC, respectively.

For a cluster heterogeneous MPSoC, we use the energy model described in Section 4.2.2, where the model parameters are given in Table 4.3 and Table 4.2. In our experiments, we use our FDM approach and the reference mapping approaches described in [CKR14] and [KYD11] to map the tasks of the streaming applications to the three MPSoCs and we compute the energy consumption of each application to

Algorithm 5: Frequency Driven Mapping

Input: A CSDF graph $G = (A, \mathcal{E})$ and a cluster heterogeneous MPSoC
Output: A task mapping for each cluster and the minimum operating frequency for each active cluster

$A^{\text{PE}}, A^{\text{EE}} \leftarrow$Apply Algorithm 2 to split all actors $\tau_i \in A$ to two parts;
if $\lceil U^{EE} \rceil > M_c^{EE} \times M_p^{EE}$ **then**
    Map some actors τ_i to PE clusters in order of decreasing utilization such that $\lceil U^{\text{EE}} \rceil \leq M_c^{\text{EE}} \times M_p^{\text{EE}}$;
if $\lceil U^{PE} \rceil > M_c^{PE} \times M_p^{PE}$ **then**
    return Unschedulable;
$\Phi_{ac} \leftarrow$Apply FFD on PE clusters and EE clusters to generate an initial task mapping and compute the frequency of each active cluster by using Equation (4.7);
`/* Remapping procedure */`
for *j=1 to* $|\phi_{ac}|$ **do**
    if $\max_{\forall \tau_i \in A_j}\{u_i\} < \frac{U_j}{M_p}$ *& an unused cluster of the same type is available* **then**
        `// U-cluster`
        $A_{j,1}, A_{j,2} \leftarrow$ Apply Algorithm 3 to split tasks;
        if $A_{j,1}$ *and* $A_{j,2}$ *can meet the condition in Proposition 2* **then**
            Keep $\forall \tau_i \in A_{j,1}$ on the initial cluster and remap $\forall \tau_i \in A_{j,2}$ to unused cluster ϕ_{unused};
            $\Phi_{ac} \leftarrow \Phi_{ac} + \phi_{\text{unused}}$;
        else
            Keep the initial mapping;
    if $\max_{\forall \tau_i \in A_j}\{u_i\} \geq \frac{U_j}{M_p}$ *& an unused cluster of the same type is available* **then**
        `// T-cluster`
        $A_{j,1}, A_{j,2} \leftarrow$ Apply Algorithm 4 to split tasks;
        if $A_{j,1}$ *can meet the condition in Proposition 3* **then**
            Keep $\forall \tau_i \in A_{j,2}$ on the initial cluster and remap $\forall \tau_i \in A_{j,1}$ to unused cluster ϕ_{unused};
            $\Phi_{ac} \leftarrow \Phi_{ac} + \phi_{\text{unused}}$;
        else
            Keep the initial mapping;
Update the operating frequencies of clusters in set Φ_{ac} by using Equation (4.7);
return Φ_{ac};

MPSoC mapping configuration using Equation (4.4), (4.5) and (4.6). The metric for the evaluation of each configuration is the energy reduction achieved by our proposed FDM approach over the different reference mapping approaches. We use the following

APP	$\|A\|$	$\|\mathcal{E}\|$	L $(cycles)$	$\mathcal{R}$(token/ $cycles$)
Beamformer	57	70	61152	1/5076
BitonicSort	40	46	2280	1/95
CHVocoder	55	70	28400	1/35550
DCT	8	7	380928	1/47616
DES	53	60	46080	1/1024
FFT	17	16	204544	1/12032
FMRadio	43	53	17208	1/1434
MP3	14	18	16795242	1/1866138
MPEG	23	26	138240	1/7680
Serpent	120	128	370296	1/3336
TDE	29	28	1071840	1/36960
Vocoder	114	147	291360	1/9105

Table 4.7: The Streaming Applications

Configuration	Granularity	PE clusters	EE clusters
MPSoC_2_20_28	2 procs	20	28
MPSoC_4_10_14	4 procs	10	14
MPSoC_8_5_7	8 procs	5	7

Table 4.8: Cluster Heterogeneous MPSoC configurations

equation to compute the energy reduction:

$$r = \frac{E_{\text{ref}} - E_{\text{FDM}}}{E_{\text{ref}}} \tag{4.17}$$

where E_{ref} is the energy consumption of an application to MPSoC mapping configuration obtained by a reference mapping approach and E_{FDM} denotes the energy consumption achieved by our proposed FDM with cluster VFS.

Comparison with [CKR14] on Heterogeneous MPSoCs

In this section, we compare our proposed FDM approach to the mapping approach proposed in [CKR14]. In [CKR14], the authors proposed several mapping approaches for cluster heterogeneous MPSoCs, and in our experiments we select the best mapping approach evaluated in [CKR14] and refer to it as CKR. In this experiment, the CKR is considered as the reference point and the energy reduction for each application

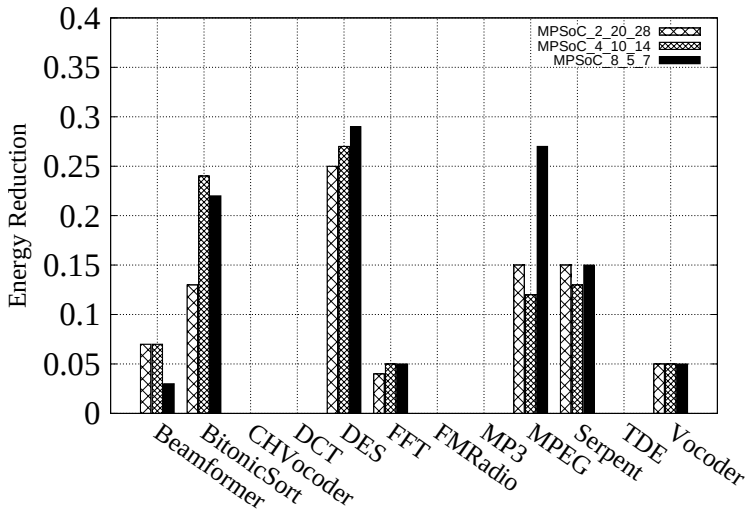

Figure 4.5: Comparison between FDM and CKR

	2 procs	4 procs	8 procs
FDM average	7%	7.7%	8.9%
Max energy reduction	25%	27%	29%

Table 4.9: Summary of Figure 4.5

benchmark is computed by using Equation (4.17). Figure 4.5 depicts the energy reduction for each benchmark mapped on the three different MPSoCs, where the x-axis shows the benchmarks and the y-axis shows the energy reduction. For 7 out of 12 benchmarks, our proposed FDM approach finds a mapping that consumes less energy compared to the one obtained by the CRK approach. The main reason is that the CKR approach is similar to the FFD algorithm but it does not consider remapping to efficiently utilize the unused clusters. Hence, for the 7 benchmarks, the remapping approach in our FDM algorithm outperforms the CKR. For the other 5 benchmarks, the remapping is not beneficial for them, so our proposed FDM approach achieves the same results as the CKR approach.

The energy reduction results are summarized in Table 4.9. We see that the average energy reduction is 7%, 7.7%, and 8.9% for the three MPSoCs with 2, 4 and 8 processors per cluster, respectively. Among all experiments, the maximum energy reduction occurs to benchmark DES which is 25%, 27%, and 29% for the three MPSoCs with 2, 4, and 8 processors per cluster, respectively.

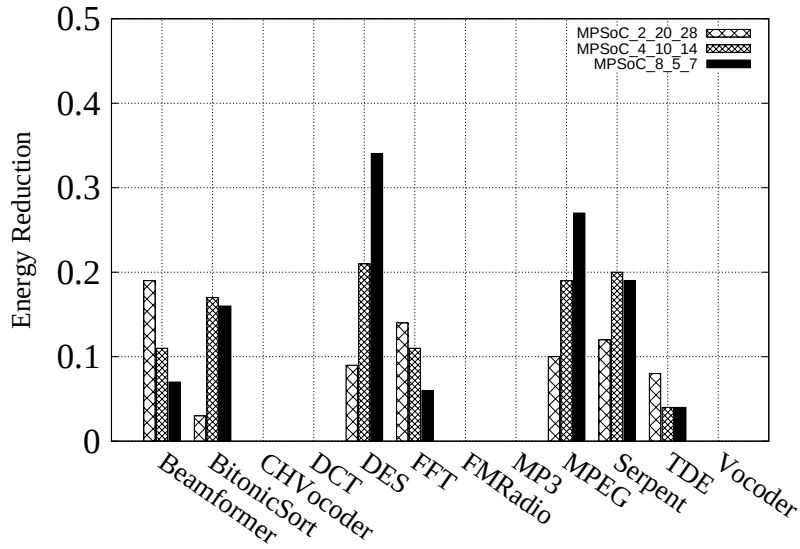

Figure 4.6: FDM vs. Algorithm 2+KYD

	2 procs	4 procs	8 procs
FDM average	6.3%	8.5%	9.4%
Max energy reduction	19%	21%	34%

Table 4.10: Summary of Figure 4.6

Comparison with [KYD11] on Heterogeneous MPSoCs

In this experiment, we compare our FDM approach with the approach proposed in [KYD11] which we refer to as KYD. Since [KYD11] only considers cluster homogeneous MPSoCs, we apply our processor type assignment proposed in Section 4.3.1, i.e., Algorithm 2, to determine the processor type for each actor and then utilize the KYD approach to map the actors to clusters. Thus, in this experiment, 'Algorithm 2+KYD' is used as the reference mapping approach in Equation (4.17).

The energy reduction for the different benchmarks mapped on the different MPSoCs is depicted in Figure 4.6. For 7 out of 12 benchmarks, our FDM finds a mapping which consumes less energy than the mapping approach 'Algorithm 2+KYD'. For the rest of the benchmarks, our proposed approach finds a mapping that consumes the same energy as the reference mapping approach. For benchmarks CHVocoder, DCT, MP3 and FMRadio, their actors which are assigned to PE type of clusters have very similar workload, hence evenly distributing heavy tasks by the KYD approach can find the energy efficient mapping as our FDM approach does. For benchmark Vocoder,

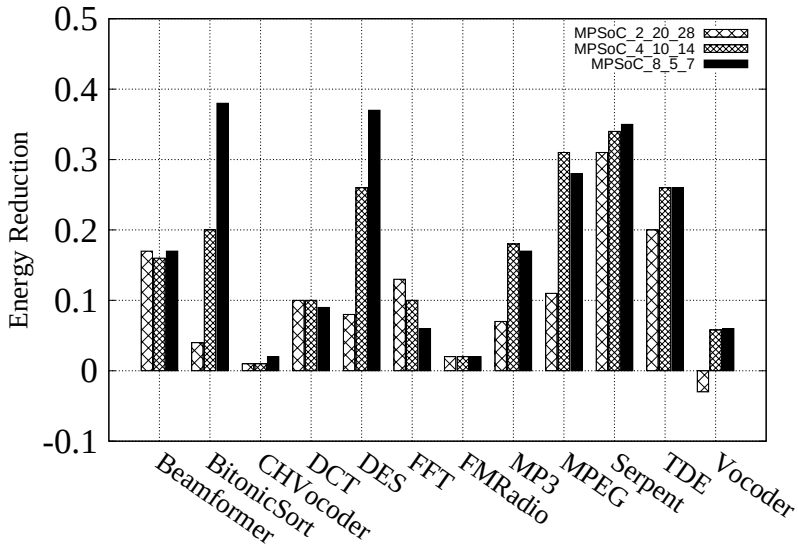

Figure 4.7: FDM vs. KYD on homogeneous MPSoCs

	2 procs	4 procs	8 procs
FDM average	10%	16.6%	18.5%
Max energy reduction	31%	34%	38%

Table 4.11: Summary of Figure 4.7

only two heavy tasks are assigned to PE clusters and running them on different clusters leads to energy efficiency, so the KYD approach can find the best mapping by mapping these two tasks to two different clusters as our FDM approach does.

The results are summarized in Table 4.10. We can see that the average energy reduction of the three MPSoCs is 6.3% for the MPSoC with 2 processors per cluster, 8.5% for the MPSoC with 4 processors per cluster, and 9.4% for the MPSoC with 8 processors per clusters. The maximum energy reduction is 19% in benchmark Beamformer for 2 processors per cluster, 21% and 34% in benchmark DES for 4 processors per cluster and 8 processors per cluster, respectively.

Comparison with [KYD11] on Homogeneous MPSoCs

The KYD approach [KYD11] is originally proposed for cluster homogeneous MPSoCs. In order to have a fair comparison, we apply our FDM approach to homogeneous MPSoCs and compare it with the KYD approach to show the efficiency of our FDM approach. Since we need to guarantee the throughput and latency constraints

shown in Table 4.7, running the benchmarks on a cluster homogeneous MPSoC comprised of EE clusters will violate the performance constraints. Thus, we only map the applications to the PE clusters available in the cluster MPSoCs described in Table 4.8, thereby considering cluster homogeneous MPSoCs that can meet the performance constraints for every application.

Figure 4.7 depicts the energy reduction of each benchmark mapped on the three different MPSoCs by only using the PE clusters. In Figure 4.7, we see that for benchmark Vocoder on platform 'MPSoC_2_20_28', the mapping derived by our FDM approach consumes 3% more energy than the KYD approach. With this fine granularity of the cluster size, i.e, the small number of processors per cluster, benchmark Vocoder has a lot of mapping possibilities. Since the KYD has a design space exploration step which enables to explore more mappings and our FDM only improves the mapping generated by the FFD heuristic, in the case of benchmark Vocoder our FDM does not find a more energy efficient mapping compared to KYD. However, except benchmark Vocoder on platform 'MPSoC_2_20_28', our FDM approach outperforms the KYD in all other cases by finding more energy efficient mappings.

The results of this experiment are summarized in Table 4.11. We see that for different MPSoCs our FDM approach can reduce the energy consumption by an average of 10%, 16.6% and 18.5%. The maximum reduction occurs for benchmark Serpent which is 31% and 34% for the MPSoCs with 2 and 4 processors per cluster, respectively. For the MPSoC with 8 processors per cluster, benchmark BitonicSort has the maximum energy reduction which is 38%.

Chapter 5

Energy Optimization for Real-Time Tasks

Di Liu, Jelena Spasic, Peng Wang, and Todor Stefanov,
"Energy-Efficient Scheduling of Real-Time Tasks on Heterogeneous Multicores Using Task Splitting",
"The 22nd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications", Daegu, South Korea, 2016, pp. 1-10.

As we discussed in Section 2.2.3, the semi-partitioned scheduling/task-splitting, e.g., [BDWZ12, GSYY10, JLBK13, BBA11], can achieve a good trade-off between global scheduling and partitioned scheduling in terms of schedulability, resource utilization and scheduling overhead. The advantage of the task-splitting technique has been extended to another dimension, namely, energy-efficient real-time scheduling. Lu and Guo [LG11] investigated to use the task-splitting technique given in [GSYY10] to energy-efficiently schedule real-time tasks under fixed-priority scheduling on homogeneous multicore systems. As presented in Section 1.1, the heterogeneous multicore systems are gradually replacing the homogeneous multicore systems in order to satisfy diverse performance requirements of different applications and at the same time reduce the energy consumption. However, there is no work investigating the task-splitting approach on heterogeneous multicore systems for the energy-efficient purpose. Motivated by this fact, in this chapter, we investigate how to adopt the task-splitting approach with dynamic priority scheduling to better utilize the resources on heterogeneous multicore systems for energy efficiency. We select the C=D approach [BDWZ12], which will be introduced in details later in Section 5.2.4, to split tasks among heterogeneous cores. We extend the C=D approach for heterogeneous multicore systems and propose an allocation algorithm to schedule real-time tasks with C=D task-splitting on heterogeneous multicore systems. Formally, our novel technical contributions are summarized as follows:

- We analyze the properties of the C=D task-splitting and extend it for heterogeneous multicore systems. We present a new definition, namely '*valid split*', for the C=D task-splitting on heterogeneous multicore systems. This analysis is presented in Section 5.4;

- Based on the analysis of the C=D task-splitting and the characteristics of heterogeneous multicore systems, we propose an energy-efficient algorithm, called ASHM, to allocate and split real-time tasks on heterogeneous multicore systems. Algorithm ASHM is presented in Section 5.5;

- Since the existing methods to compute the minimum operational frequency for each core cannot work with the C=D approach, we propose a new approach based on Quick convergence Processor-demand Analysis (QPA) [ZB09] to compute the minimum frequency for each core in the system. This proposed approach is presented in Section 5.5.4

The extensive experiments on synthetic real-time tasks shows the effectiveness of our ASHM algorithm over the existing partitioned algorithms in terms of energy efficiency.

5.1 Related Work

Energy-efficient scheduling for real-time systems has been widely explored in the past two decades. Chen and Kuo in [CK07] comprehensively reviewed most of the papers addressing energy-efficient real-time scheduling problems before 2007. An updated review for energy-efficient real-time scheduling is provided by Bambagini et al. in [BMAB16]. We can see from [CK07, BMAB16] that most of the works consider homogeneous systems, whereas in this work we consider heterogeneous multicore systems which are more energy-efficient but more difficult to effectively schedule the tasks.

A few works consider heterogeneous systems. Chen and Thiele in [CT08] proposed a polynomial algorithm to energy efficiently schedule periodic tasks on heterogeneous systems but the systems they considered had only two cores. In contrast, we consider a more general system model where the system has two types of cores, and for each core type we can have any number of cores which can be seen on many real commercial processors. Chen *et al.* [CST09] developed two polynomial-time algorithms to energy efficiently allocate real-time tasks on a more general system model that can have different types of processors and different number of processors for each type like we consider in our work. However, in their work, they do not take voltage/frequency scaling (VFS) into account, whereas we consider VFS as a crucial

technique to improve the energy efficiency. With the consideration of VFS, we can further minimize the energy consumption of the heterogeneous system. In [HTC07], Huang *et al.* proposed an allocation algorithm to schedule frame-based real-time tasks on heterogeneous multicore systems, where a non-preemptive scheduling is considered. The main difference compared to our work is: (1) they consider frame-based real-time task model, whereas we consider the periodic task model which is more general; (2) the non-preemptive scheduling, they consider, is known to be NP-hard in the strong sense even on uniprocessor [JSM91]. In contrast, we consider preemptive scheduling.

Recently, more interests have risen for energy efficient real-time scheduling on single-ISA heterogeneous multicore systems. Liu *et al.* [LSCS15] consider an optimal cluster scheduling to schedule real-time tasks on cluster heterogeneous multicore systems. However, from practical perspective the optimal cluster scheduling suffers from a very high overhead caused by frequent context switching and task migration. When the overhead is taken into account, the achieved resource utilization may be quite low in practice [BBA11]. In contrast, the C=D task-splitting, we consider, has a limited number of migrations and on each core a normal EDF scheduler is used to schedule real-time tasks, hence it significantly reduces the context-switching and task migration overhead and makes it more practical for real implementation. Colin *et al.* [CKR14] and Elewi *et al.* [ESAS14] adopt the partitioned EDF scheduling to schedule real-time tasks on heterogeneous multicore systems, where both consider energy minimization as the objective. Due to the capacity loss of partitioned scheduling, the proposed approaches from [CKR14] and [ESAS14] do not fully utilize 'LITTLE' cores on a heterogeneous multicore system and thus possibly lose some opportunities to further reduce the energy consumption. Contrarily, in our work, we adopt the state-of-the-art C=D task-splitting approach to exploit the energy efficiency of a heterogeneous multicore system. Our experimental results on randomly generated task sets demonstrate the merit of the task-splitting on heterogeneous multicore systems.

A few works study the task migration/splitting approaches for energy-efficient real-time multicore system. Chen *et al.* [CHC$^+$04] address the energy-efficient scheduling problem on homogeneous multicore systems with task migration, in which all tasks have the same release time and a common deadline. In our work, we consider a more general and widely-used periodic task model and instead of homogeneous multicore systems, we consider heterogeneous multicore systems which are more energy efficient. Lu and Guo [LG11] adopt the task-splitting approach proposed by Guan *et al.* [GSYY10] on homogeneous multicore systems to achieve energy efficiency. The main difference between [LG11] and our work is twofold:1) they consider fixed priority scheduling, whereas dynamic priority scheduling, i.e., earliest deadline first (EDF) [LL73], is adopted in our work. It is known that EDF can achieve better re-

source utilization than fixed-priority scheduling; 2) they consider homogeneous multicore systems, whereas we target heterogeneous multicore systems which are more energy-efficient.

5.2 Background

In this section, we present the system model, task model, and energy model used in this work. Then, we give a brief description of the C=D task-splitting approach [BDWZ12].

5.2.1 System Model

We consider a heterogeneous multicore system M which consists of two types of cores, the 'big' core for performance and the 'LITTLE' core for low power. Throughout this chapter, we use PE and EE to denote a 'big' core and a 'LITTLE' core, respectively, like what we did in Chapter 4. We use M_{EE} and M_{PE} to denote the sets consisting of all EE cores and all PE cores, respectively. The power consumption of one core can be computed by the following equation,

$$P(f) = \alpha f^b + s \tag{5.1}$$

where α and $b \in [2, 3]$ are technology-based parameters [CK07], f is the operational frequency. For different types of cores, α and b are different. The first term of Equation (5.1) is the frequency-related power consumption, i.e., the dynamic power consumption. s denotes the power consumption unrelated to the frequency, i.e., the static power consumption. Each core executes independently from the others and has a discrete frequency set at which the core can run. Let $\overline{f_j} = \{f_1, \cdots, f_l\}$ denote the frequency set of core j. Without loss of generality, we assume that the frequencies in the set are sorted in increasing order, i.e., $f_k < f_{k+1}$.

5.2.2 Task Model

The task model adopted in this work is similar to the one introduced in Section 2.2.1, but all tasks are assumed to start at time instant 0, i.e., $S_1 = S_2 = \cdots = S_n = 0$. Moreover, since we have two type of cores, the WCET of each task may vary when executing on different types of cores. We slightly extend the model to have two WCETs as we did in Chapter 4.

- C_i^{EE} and C_i^{PE} are the worst-case execution times (WCETs) of task τ_i executing on an EE core and PE core at the maximum frequency, respectively;

Then, a task is characterized by a tuple of parameters $\tau_i = \{C_i^{EE}, C_i^{PE}, D_i, T_i\}$.

5.2.3 Energy Model

With the system and task models discussed above, we explain how to compute the energy consumption for the system. After all tasks are allocated to cores, the energy consumption for each core can be computed as follows:

$$E_j = hp\left(\alpha_j f_j^{b_j} \frac{f_{max}}{f_j} \sum_{\forall \tau_i \in \Gamma_j} \frac{C_i}{T_i} + s_j\right) \tag{5.2}$$

where Γ_j is the task set containing all tasks allocated to core j and hp is the hyper-period of task set Γ_j. The hyper-period is the least common multiple (*lcm*) of all tasks' periods. Every hyper-period has the same workload and thus we compute the energy consumption within one hyper-period. The energy consumption of the whole system is the summation of the energy consumption E_j of all cores.

5.2.4 C=D Task-Splitting

In this work, we adopt the C=D task-splitting to schedule real-time tasks on a heterogeneous multicore system. Burns *et al.* in [BDWZ12] propose the C=D approach to split real-time tasks on homogeneous systems. They use a preemptive earliest deadline first (EDF) scheduling [LL73] to schedule the tasks on each core. The tasks are first allocated to cores according to a certain allocation algorithm. If task τ_i cannot be integrally allocated to a core, the C=D approach splits unassigned task τ_i into two parts/subtasks, τ_i^1 and τ_i^2. The split procedure is as follows:

- Find a processor x and then compute the maximum computation time C_i^1 for subtask τ_i^1 which ensures the schedulablility of subtask τ_i^1 on processor x. For subtask τ_i^1, its deadline D_i^1 is set to be equal to C_i^1 and its period T_i^1 is equivalent to its original period T_i, i.e., $\tau_i^1 = \{C_i^1, D_i^1 = C_i^1, T_i^1 = T_i\}$. Then, subtask τ_i^1 is allocated to processor x;

- According to subtask τ_i^1, we can obtain the second subtask τ_i^2. The WCET C_i^2 of τ_i^2 is computed as $C_i^2 = C_i - C_i^1$, its deadline D_i^2 is computed as $D_i^2 = D_i - D_i^1$ and its period T_i^2 equals to its original period, $T_i^2 = T_i$, i.e., $\tau_i^2 = \{C_i^2 = C_i - C_i^1, D_i^2 = D_i - D_i^1, T_i^2 = T_i\}$. Subtask τ_i^2 is allocated to a processor which has enough space to schedule subtask τ_i^2 and is different from processor x on which subtask τ_i^1 is allocated.

In the remainder of this chapter, we call subtask τ_i^1 the first subtask and subtask τ_i^2 the second subtask.

The C=D task-splitting permits each core to have only one first subtask τ_i^1. This means that the whole system has at most M split tasks, where M is the number of

cores. This task-splitting scheme can be realized by using task migration. τ_i^1 completes its execution on the allocated core. Then it migrates to the core where τ_i^2 is assigned and continues the execution of subtask τ_i^2. From the experimental results in [JLBK13], the C=D task-splitting outperforms other existing semi-partition/task-splitting approaches in terms of schedulability.

Migration Overhead: Like [BDWZ12], in our work the migration overhead is assumed to be negligible. An extensive number of experiments on real hardware systems [BBA11] have shown that with cache coherence among cores the task migration overhead is at the similar order of magnitude as the normal context switching. The cache coherence hardware architecture, like CoreLink CCI-400 Cache Coherent Interconnect [ARM16], has been adopted by the big.LITTLE multicore systems to maintain the cache coherence between cores. Therefore, the migration overhead is accounted for in the WCET of a task.

5.3 Motivational Example

In this section, we use an example to motivate the application of the C=D task-splitting approach on heterogeneous multicore systems for energy efficiency purpose. For simplicity, assume that we have a multicore system with one PE core and one EE core. The PE core and EE core have different power parameters α and b (see Equation (5.1) and (5.2)). In this example, we use the parameter values from Table 4.3 given in Section 4.2.2 of Chapter 4. We recap it here for reference convenience.

Core type	$\alpha(W/Mhz^b)$	b	$s(W)$
PE	3.03×10^{-9}	2.621	0.155
EE	2.62×10^{-9}	2.12	0.027

Table 5.1: Power parameters for different core types

	$C^{PE}(ms)$	$C^{EE}(ms)$	D(ms)	T(ms)
τ_1	55	110	100	100
τ_2	20	40	100	100
τ_3	20	40	100	100
τ_4	15	30	100	100

Table 5.2: The original task set

	$C^{PE}(ms)$	$C^{EE}(ms)$	D(ms)	T(ms)
τ_4^1	10	20	20	100
τ_4^2	5	10	80	100

Table 5.3: Split subtasks

Suppose to have four tasks with the parameters given in Table 5.2. As far as the deadlines can be ensured, we strive to partition/allocate as many tasks as possible to the EE core in order to save energy consumption. However, since scheduling τ_1 on the EE core will violate the deadline guarantee, only τ_2, τ_3, and τ_4 are eligible to be scheduled on the EE core. But we cannot schedule τ_2, τ_3, and τ_4 together on the EE core, because a total utilization of $1.1 > 1$ leads to infeasibility. One task has to be scheduled on the PE core along with τ_1. Then, we obtain a fully partitioned allocation for the given task set, where τ_1 and τ_4 are scheduled on the PE core and τ_2 and τ_3 are scheduled on the EE core. In contrast to the above fully partitioned allocation, we adopt the C=D task-splitting (explained in Section 5.2.4) to schedule the tasks on the multicore system. In the splitting case, τ_1 is scheduled on the PE core while τ_2 and τ_3 are scheduled on the EE core. But τ_4 is split into two subtasks, τ_4^1 and τ_4^2, and then we schedule τ_4^1 on the EE core and τ_4^2 on the PE core. The parameters for the subtasks are shown in Table 5.3,

With the given allocation and the power parameters, we can compute a minimum frequency for each core such that the energy consumption can be minimized by using VFS while deadlines are still ensured. Table 5.4 shows the allocation, the minimum operational frequency of each core, and the energy consumption of the multicore system. We can see that the splitting approach saves energy consumption by 32% compared to the partitioned approach because it can effectively utilize the EE core to save energy and at the same time it can reduce the workload allocated to the PE core. As a result, the PE core in the splitting approach executes at a lower frequency compared to the partitioned approach.

Mapping	PE	EE	f^{PE}	f^{EE}	Energy(mJ)
Partitioned	τ_1, τ_4	τ_2, τ_3	$1.4GHz$	$1.2GHz$	5.42
Splitting	τ_1, τ_4^2	τ_2, τ_3, τ_4^1	$1.2GHz$	$1.4GHz$	3.69

Table 5.4: Energy consumption

From the example, we see the advantage of the C=D task-splitting approach on heterogeneous systems in terms of energy efficiency. In the subsequent sections, we will introduce our novel approach to exploit the C=D task-splitting on heterogeneous

multicore systems for minimizing the energy consumption.

5.4 C=D Task-Splitting on Heterogeneous Multiprocessor Systems

In [BDWZ12], the C=D task-splitting is devised for homogeneous multiprocessor systems. However, in our work, we target heterogeneous multicore systems [Mit15][Mit16] which have been emerging as an alternative of the conventional homogeneous multicore systems. In this section, we investigate how to adopt the C=D task-splitting on a heterogeneous system.

5.4.1 Task Splitting

	C^{PE}	C^{EE}	D	T
τ_1	60	120	100	100
τ_1^1	25	50	50	100
τ_1^2	35	70	50	100
τ_1^1	41	82	82	100
τ_1^2	19	38	18	100

Table 5.5: Let us assume that we split τ_1 into two subtasks τ_1^1 and τ_1^2 and allocate τ_1^1 and τ_1^2 to an EE core and a PE core, respectively. We assume that there is no constraint on the split. We give two different splits for τ_1 shown in rows 3,4 and 5,6. For the first split shown in rows 3,4, there is no problem to schedule the subtasks. However, for the second split, although the execution time on the EE core is maximized, it causes a deadline miss for subtask τ_1^2 due to $C^{PE} > D$, seen in the last row with red color.

Since, on heterogeneous multicore systems, a task's WCET is varying upon the allocated core, the splitting on the heterogeneous multicore system should pay more attention to the varying WCET and the relation between the obtained two subtasks. First, the deadline of the first subtask τ_i^1 is set according to where the first subtask is allocated. For instance, assume that a subtask τ_i^1 has its $C_i^{PE} = 5$ and $C_i^{EE} = 10$. If it is allocated to a PE core, its deadline D_i^1 equals to $C_i^{PE} = 5$, otherwise $D_i^1 = C_i^{EE} = 10$ if allocated to an EE core. Moreover, in some cases an improper split might cause a deadline miss for the second subtask τ_i^2. The example given in Table 5.5 demonstrates this issue.

From the example, we observe the potential split issue on a heterogeneous multicore system. Thus, we give the following property to ensure that a proper split on

heterogeneous multicore systems is obtained:

Property 1. *On a heterogeneous multicore system, the following inequality must hold for a split task τ_i,*

$$T_i - C_i^1 \geq C_i^2 \tag{5.3}$$

where C_i^1 and C_i^2 are the WCETs of subtasks τ_i^1 and τ_i^2, depending on which type of core the subtasks have been allocated.

This property is to ensure enough space to execute the second subtask τ_i^2 on a heterogeneous system. We can see that for subtask τ_i^2 it must have,

$$D_i^2 \geq C_i^2 \tag{5.4}$$

Since $D_i^2 = D_i - D_i^1 = T_i - D_i^1$ and $D_i^1 = C_i^1$, see Section 5.2.4, we obtain

$$T_i - C_i^1 \geq C_i^2 \tag{5.5}$$

Thus, the property is observed. Based on this property, we give the following definition,

Definition 5.4.1 (valid split)**.** If two subtasks τ_i^1 and τ_i^2 obtained by splitting task τ_i satisfy Property 1, we call such split *a valid split*.

If the split is not a *valid split*, then the second subtask cannot meet its deadline.

5.4.2 Subtask Allocation

In Section 5.4.1, we discussed how to find a *valid split* for a task on a heterogeneous multicore system. Here, we continue to discuss the allocation of subtasks. Before proceeding to the discussion, we distinguish tasks in two categories and give their definitions as follows,

Definition 5.4.2. If a task can be integrally scheduled on an EE core, we call such task an **eligible task** (E-task).

Definition 5.4.3. If a task **cannot** be integrally scheduled on an EE core, we call such task a **non-eligible task** (NE-task).

If we look at the motivational example in Section 5.3-Table 5.2, τ_2, τ_3, and τ_4 are E-tasks and τ_1 is NE-task. Now, we discuss the possible allocation destinations for these two categories of tasks.

E-task

When an E-task is selected to be split, any split is a *valid split* regardless of which type of core the subtasks are allocated. Therefore, for an E-task, the two subtasks can be allocated to any type of core, as long as the schedulability of the system is ensured. Thus, we can have three possible combinations to allocate the two subtasks of an E-task:

- Allocate the two subtasks to two EE cores;
- Allocate the two subtasks to one EE core and one PE core; and
- Allocate the two subtasks to two PE cores.

NE-task

When a NE-task is about to be split, we need to ensure that the obtained split is a valid split by satisfying Property 1. For a NE-task, we cannot allocate the two subtasks to two EE cores, because Property 1 will be violated and then it leads to an invalid split. Excluding the invalid combination, we have two possible combinations to allocate the two subtasks of a NE-task:

- Allocate the two subtasks to one EE core and one PE core; and
- Allocate the two subtasks to two PE cores.

With the above possible allocation destinations for the two categories of tasks, in the next section, we will use this information to devise an energy-efficient allocation strategy for each category of tasks.

5.5 Allocation and Split on Heterogeneous Multicore Systems (ASHM)

In [CT08], Chen and Thiele have shown that allocating real-time tasks onto two different processors is an NP-hard problem. Their problem is just a subset of our problem, so our problem is also an NP-hard problem. Hence, we propose a heuristic algorithm to energy-efficiently schedule real-time tasks on heterogeneous multicore systems with task-splitting. We call this algorithm ASHM. ASHM first handles all E-tasks and then all NE-tasks. For the sake of clarity, we first explain the different parts in the ASHM algorithm and after that we explain the whole ASHM algorithm. Before proceeding to the detailed discussion, we introduce the following property for the core with first subtask τ_i^1 allocated on it,

Property 2. *A core must run at the maximum frequency if first subtask τ_i^1 of a split task τ_i is assigned to it.*

It is trivial to see this property because the first subtask of a split task has its WCET equal to the deadline. Scaling down the frequency leads to a deadline miss. This property is useful to determine the allocation of the subtasks.

5.5.1 Allocation and splitting of E-tasks

ASHM first starts to allocate and split E-tasks. The procedure to allocate and split E-tasks is summarized as follows:

1. Use a bin-packing algorithm, first-fit-decreasing (FFD) [CGJ97], to integrally allocate E-tasks to EE cores;

2. Split unallocated E-tasks on the platform. For a given unallocated E-task τ_i, we use the following allocation and splitting order,

   (a) Split τ_i among two EE cores. If it fails, try step b);

   (b) Split τ_i among one EE core and one PE core. If fails, try step c);

   (c) Allocate τ_i integrally to one PE core. If it fails, try step d);

   (d) Split τ_i among two PE cores. If it fails, the system is unschedulable on the platform with $M = \{M_{EE}, M_{PE}\}$.

For the first step, we use FFD to integrally allocate EE tasks to EE cores because FFD is proven to be the resource efficient bin-packing algorithm [AY03]. By using FFD we could leave some EE cores with a lot of free capacity. This could later benefit the NE-tasks for energy saving.

After some E-tasks are integrally allocated to EE cores, we might have some E-tasks left unallocated. The next step is to split and allocate them on the system. The allocation and split order summarized above prioritizes the EE cores to explore the energy-efficient potential on the EE cores. Therefore, we first try to allocate the subtasks of a split E-task to two EE cores. If the task cannot be split among two EE cores, this means that there is no enough space on EE cores. So, we try one EE core and one PE core. Since, a PE core consumes much more power than an EE core and Property 2 indicates the maximum frequency requirement, it is not favorable to allocate the first subtask to a PE core. Therefore, we constrain ourself to allocate the first subtask to an EE core and the second subtask to a PE core. For the selection of the PE cores, we use the approach proposed in [CKR14] which selects the core with the smallest energy cost contribution to the whole system when the task is allocated to it. If the

combination of one EE core and one PE core still fails, we need to find an allocation among PE cores.

On PE cores, we first try to integrally allocate the E-task to one PE core because if we split an E-task among two PE cores, Property 2 requires that one PE core must execute at the maximum frequency which leads to a very high power consumption. Hence, we prefer to integrally allocate the E-task to one PE core than split it among two PE cores. We also use the approach from [CKR14] to select the energy-efficient core for the task. If it still fails, we try the final step to split it on two PE cores in order to ensure its schedulability.

Algorithm 6 presents the pseudo-code to allocate and split E-tasks, called EAS, following the procedure explained above. EAS takes as inputs task set Γ_E consisting of all E-tasks and the heterogeneous multicore platform consisting of EE core set M_{EE} and PE core set M_{PE} and outputs the allocation of all E-tasks. At Line 1, we first use FFD to allocate E-tasks to EE cores integrally. If there are some unallocated E-tasks, we follow the steps introduced above to split unallocated E-tasks among two EE cores or one EE core and one PE core - see Line 3-10. We use function mpwr() to represent the core selection approach from [CKR14], where the inputs of mpwr() are a core set and a task and the output is a core which can schedule the task and has the smallest contribution to the energy consumption. However, if the task is not allocated successfully, we have to try to allocate or split the task among PE cores - see Line 11-24. From Line 12-14, the integral allocation on one PE core is first tried. If it fails, from Line 15-24 EAS splits τ_i among two PE cores. Function SPLIT in Algorithm 6 finds the first subtask τ_i^1 with the maximum WCET which is schedulable on core x and also gives the corresponding τ_i^2. We will explain SPLIT in details later in Section 5.5.3.

5.5.2 Allocation and Splitting of NE-tasks

After all E-tasks are allocated, we proceed towards allocating and splitting NE-tasks on the system. The procedure to allocate and split NE-task τ_i is summarized as follows:

1. Split τ_i among one EE core and one PE core. If it fails, try step 2);
2. Allocate τ_i integrally onto one PE core. If it fails, try step 3);
3. Split τ_i among two PE cores. If it fails, it is unschedulable.

Since, after the allocation of E-tasks, EE cores might have some free space to execute parts of NE-tasks, we first try to split a NE-task among one EE core and one PE core in order to utilize EE cores for energy saving. Since the first subtask needs a

Algorithm 6: E-task Allocation and Split (EAS)

Input: All E-tasks Γ_E and the heterogeneous multicore platform $M = \{M_{EE}, M_{PE}\}$
Output: Allocation for all E-tasks

```
1  M_EE ← using FFD to allocate tasks from Γ_E
2  Γ_un ← unallocated tasks from Γ_E
3  for ∀τ_i ∈ Γ_un in order of decreasing U do
4      for ∀x ∈ M_EE in order of increasing U do
5          τ_i^1,τ_i^2 = SPLIT(τ_i, x)
6          if τ_i^1 ≠ ∅ then
7              x ← τ_i^1
8              y ← mpwr(M = {M_EE, M_PE}, τ_i^2)
9              if y = ∅ then
10                 x ← x − τ_i^1
11     if τ_i is not allocated successfully then
12         x ← mpwr(M_PE, τ_i)
13         if x ≠ ∅ then
14             x ← τ_i
15         else
16             for ∀x ∈ M_PE in order of decreasing U do
17             τ_i^1,τ_i^2 = SPLIT(τ_i, x)
18             if τ_1^1 ≠ ∅ then
19                 x ← τ_i^1
20                 y ← mpwr(M_PE, τ_i^2)
21                 if y = ∅ then
22                     x ← x − τ_i^1
23                 else
24                     y ← τ_i^2
25     if τ_i is not allocated successfully then
26         return Unschedulable
27 return Allocation of ∀τ_i ∈ Γ_E
```

maximum operational frequency (Property 2), we constrain the first subtask to the EE core and allocate the second subtask to a PE core for ensuring the schedulability. However, when we maximize the execution time of the first subtask on an EE core, it might bring a negative effect on the second subtask. Maximizing the execution of the first subtask will reduce the slack time for the second subtask, i.e., $D_i^2 - C_i^2$. As a consequence, the reduced slack time leaves a little space to scale down the frequency of the PE core which might compromise the energy saving from the EE core. Hence, in order to provide an energy-efficient split, we set the following constraint for splitting a NE-task on one EE core and one PE core.

$$\frac{C_i^2}{D_i^2} \leq \frac{C_i}{T_i} \tag{5.6}$$

Constraint (5.6) can guarantee that after the split the slack ratio of the second subtask is not smaller than before. Therefore, it would not require to run at a higher frequency. If the task cannot be split on one EE core and one PE core, we integrally allocate NE-task τ_i to one PE core. For the integral allocation, we try to allocate task τ_i to the PE core given by function mpwr(). If task τ_i cannot be allocated to a PE core, then we split it among two PE cores in order to ensure its schedulability.

Algorithm 7 presents the pseudo-code to allocate and split NE-tasks, where we call this algorithm NEAS. The inputs for NEAS are all NE-tasks and the platform. From Line 2-13, NEAS splits task τ_i among one EE core and one PE core. For this combination NEAS selects the EE core with the smallest utilization and the PE core given by function mpwr() to split task τ_i in order to save the energy consumption. At Line 5-7 constraint (5.6) is checked. If the combination of one EE core and one PE core fails to allocate task τ_i, then, from Line 14-17, NEAS tries to integrally allocate task τ_i to one PE core which can schedule τ_i and has the minimum contribution to the energy consumption. If it does not successfully allocate τ_i to one PE core, NEAS splits τ_i among two PE cores from Line 18-24. In this case, it finds the PE core with the largest utilization to schedule the first subtask τ_i^1. Because τ_i^1 requires the maximum frequency to guarantee the schedulability and the PE core with the largest utilization should execute at a high frequency compared to others, allocating τ_i^1 to the PE core would not increase the frequency too much which in turn does not lead to a lot of extra energy consumption for the task allocated to the PE core. For τ_i^2, we still use function mpwr() to find the candidate core. If splitting among two PE cores fails, NEAS returns a failure.

5.5.3 The SPLIT function

In this section, we present the SPLIT function used in EAS and NEAS discussed above. Algorithm 8 presents the pseudo-code for SPLIT. The concept behind the SPLIT algo-

Algorithm 7: NE-task Allocation and Split (NEAS)

Input: All NE-tasks Γ_{NE} and the heterogeneous multicore platform $M = \{M_{EE}, M_{PE}\}$
Output: Allocation for all NE-tasks

for $\forall \tau_i \in \Gamma_{NE}$ *in order of decreasing* U **do**
  for $\forall x \in M_{EE}$ *in order of increasing* U **do**
    τ_i^1,τ_i^2 = SPLIT(τ_i, x)
    if $\tau_i^1 \neq \emptyset$ **then**
      while $\frac{C_i^2}{D_i^2} > \frac{C_i}{T_i}$ **do**
        $C_i^1 \leftarrow C_i^1 - 1$
        Recompute C_i^2 according to the new C_i^1 (see Section 5.2.4)
      $x \leftarrow \tau_i^1$
      $y \leftarrow \mathsf{mpwr}(M_{PE}, \tau_i^2)$
      if $y = \emptyset$ **then**
        $x \leftarrow x - \tau_i^1$
      else
        $y \leftarrow \tau_i^2$
  if τ_i *is not allocated* **then**
    $y \leftarrow \mathsf{mpwr}(M_{PE}, \tau_i^2)$
    if $y \neq \emptyset$ **then**
      $y \leftarrow \tau_i$; break
    for $\forall pe \in M_{PE}$ *in order of increasing* U **do**
      τ_i^1,τ_i^2 = SPLIT(τ_i, pe)
      if $\tau_i^1 \neq \emptyset$ **then**
        $x \leftarrow \tau_i^1$
        $y \leftarrow \mathsf{mpwr}(M_{PE}, \tau_i^2)$
        if $y = \emptyset$ **then**
          $pe \leftarrow pe - \tau_i^1$
  if τ_i *is not allocated successfully* **then**
    return Unschedulable
return Allocation of $\forall \tau_i \in \Gamma_{NE}$

Algorithm 8: SPLIT

Input: τ_i and one processor x
Output: subtasks τ_i^1, τ_i^2

$C_i^1 = D_i^1 = (0.999 - U_x)T_i$;
Compute subtask τ_i^2 according to the parameters of τ_i^1 (see Section 5.2.4)
while $C_i^2 > T_i - C_i^1$ *and* τ_i *is a NE-task and* x *is an EE core* **do**
    $C_i^1 \leftarrow C_i^1 - 1$
    Recompute C_i^2 according to the new C_i^1 (see Section 5.2.4)
$\Gamma_x \leftarrow \Gamma_x + \tau_i^1$;
while *True* **do**
    if $C_i^1 < 1$ **then**
        return $\tau_i^1 = \tau_i^2 = \emptyset$
    if *QPA(*Γ_x*) reports unschedulable* **then**
        $t \leftarrow$the failure point from QPA
        while *True* **do**
            $I = (t - \text{dbf}(\Gamma_x - \tau_i^1, t))/\lfloor \frac{t+T_i-(C_i^1-1)}{T_i} \rfloor$
            if $I \neq C_i^1$ **then**
                $C_i^1 = C_i^1 - 1$
            else
                Break;
    else
        Compute parameters for subtask τ_i^2 (see Section 5.2.4)
        return τ_i^1, τ_i^2

rithm is based on the approach proposed in [BDWZ12] and the properties of the C=D approach on heterogeneous multicore systems identified and discussed in Section 5.4. The inputs for SPLIT are a task τ_i and a core x while the output is two subtasks τ_i^1 and τ_i^2. The objective of function SPLIT is to find the maximum WCET of τ_i^1 which can satisfy the schedulability on core x. The procedure is as follows:

- Initialize the parameters of subtasks. For τ_i^1 let $C_i^1 = D_i^1 = (0.999 - U_x)T_i$ (Line 1) and configure subtask τ_i^2 according to subtask τ_i^1 (Line 2), as explained in Section 5.2.4, where U_x denotes the total utilization of processor x;
- If τ_i is a NE-task and x is an EE core (Line 3-5), ensure valid split according to Property 1;
- Use QPA [ZB09] to test whether subtask τ_i^1 can be allocated onto core x. If it is schedulable, return the subtasks τ_i^1 and τ_i^2 (Line 10, 18-20);
- If QPA reports 'unschedulable', recompute the WCET for subtask τ_i^1. In this case, we use the recurrence approach from [BDWZ12] to make sure that

$$C_i^1 = (t - \text{dbf}(\Gamma_x - \tau_i^1, t))/\lfloor \frac{t + T_i - (C_i^1 - 1)}{T_i} \rfloor \tag{5.7}$$

  where t is the failure point returned by QPA, i.e., the time instance $dbf(\Gamma_x, t) > t$ and $\text{dbf}(\Gamma_x - \tau_i^1, t)$ represents the demand of tasks on core x excluding subtask τ_i^1. The recurrence equation in Equation (5.7) computes a maximum value for C_i^1 such that $\text{dbf}(\Gamma_x, t) \leq t$ which ensures the schedulability of task set Γ_x at time instant t. If Equation (5.7) is satisfied, the recurrence procedure stops and returns C_i^1 for subtask τ_i^1. Otherwise, it decrements C_i^1 by 1 and repeats the previous procedure (Line 10-17);
- Return failure if it cannot split task τ_i on core x (Line 8-9).

Note that we use 0.999 instead of 1 to initialize a subtask at Line 1, because if using 1 would result in that QPA uses the hyper-period of all tasks as bound to test the schedulability. Then, QPA would be very complex and time-consuming.

5.5.4 Computing the minimum frequency

We use VFS to scale down the frequency of each core so that the energy consumption is further reduced. However, next to implicit deadline tasks (i.e., unsplit tasks), we might have some subtasks obtained by splitting on some cores which are constrained deadline tasks. In such case, we cannot simply use the utilization-based approach [CK07] [BMAB16] to compute the minimum frequency. Hence, we integrate the

Algorithm 9: Compute Minimum Frequency (CMF)

Input: core x and task set Γ_x
Output: the minimum operating frequency for core x

```
1 if x has a first subtask then
2     return f_max
3 else
4     Compute a minimum achievable frequency f_crit based on U_x
5     f̄ ← {∀f_i | f_i ≥ f_crit} and sort f̄ in order of increasing frequency
6     for ∀f_i ∈ f̄,   i = {1, 2, ..., k} do
7         if QPA(Γ_x, f_i) reports schedulable then
8             return f_i
9     return f_max
```

frequency into QPA [ZB09] to efficiently compute the minimum frequency for a core.

Algorithm 9 (CMF) presents the pseudo-code to compute the minimum operational frequency for each core. The inputs are one core x and a task set Γ_x which includes all tasks allocated to core x. The output is the minimum operational frequency for core x. If the core has first subtask τ_i^1, its frequency will be set to the maximum frequency according to Property 2 - see Line 1-2. Otherwise, we compute a minimum operational frequency for the core from Line 4-8. First, we compute a frequency called f_{crit} based on utilization U_x of core x [CK07]. Frequency f_{crit} can be deemed as the lower bound of the operational frequency of core x. If the operational frequency is lower than f_{crit}, the system is not schedulable. Then, we select all frequencies from the core's frequency set which are greater than f_{crit} and let these frequencies form a frequency set $\overline{f}$ sorted in order of increasing frequency - see Line 5. We start with the smallest frequency f_i in frequency set $\overline{f}$ and use QPA to test whether the task set is schedulable at this frequency - see Line 7. If it is schedulable, CMF returns frequency f_i as the operational frequency. Otherwise, we take frequency f_{i+1} and use QPA to test whether the task set is schedulable at this frequency.

5.5.5 The ASHM Algorithm

Given all algorithms explained earlier, we present our complete Allocation and Split algorithm ASHM using the pseudo-code in Algorithm 10. We first divide all tasks into two task sets Γ_E and Γ_{NE}, one for all E-tasks Γ_E and another for all NE-tasks Γ_{NE}. Then, we use EAS (Algorithm 6) to allocate all E-tasks - see Line 2. If all E-

Algorithm 10: ASHM

Input: all tasks Γ and the platform $M = \{M_{EE}, M_{PE}\}$
Output: the allocation for all tasks and the minimum operational frequency for each core on the platform

1 $\Gamma_E \leftarrow$all E-tasks, $\Gamma_{NE} \leftarrow$all E-tasks
2 $M \leftarrow$ EAS(Γ_E, M)
3 $M \leftarrow$ NEAS(Γ_{NE}, M)
4 **for** $\forall x \in M$ **do**
5 $\quad f_x \leftarrow$ CMF(x, Γ_x)

tasks are successfully allocated, we proceed to allocate all NE-tasks by using NEAS (Algorithm 7) - see Line 3. Finally, we apply CMF (Algorithm 9) to compute the minimum frequency for each core - see Line 4-5.

Complexity Analysis: In the worst case, EAS, NEAS, SPLIT and CMF are all pseudo-polynomial algorithms due to QPA. Although QPA has shown its efficiency in [ZB09], its complexity is still pseudo-polynomial in the worst case. This worst-case scenario happens when the utilization U equals to 1. However, in function SPLIT, we strive to avoid the worst-case scenario to occur by setting the utilization bound as 0.999 - see Line 1 of Algorithm 8. Therefore, in practice, our algorithms can be executed very efficiently.

5.6 Evaluation

In this section, we present extensive experimental results to show the effectiveness of our ASHM algorithm in terms of energy consumption compared to two widely-used bin-packing algorithms [CGJ97] and two existing related approaches [CKR14] [ESAS14]. We do not compare with the algorithm proposed in Chapter 4, because the approach proposed in Chapter 4 is very similar to [CKR14] when we consider the per-core VFS system. We do not compare with [CST09], because they do not take VFS into account. Therefore, our approach will always save more energy consumption than [CST09]. Since the authors in [CKR14] have shown that their approach outperforms the allocation approach proposed in [HTC07], we do not compare our ASHM to [HTC07].

5.6.1 Experimental Setup

Task Generation

To evaluate the effectiveness of ASHM, we adopt the widely-used random task generator based on *UUnifast-discard* [DB11]. *UUnifast-discard* enables the generation of unbiased task sets. It takes as inputs the number of tasks n and the total utilization U and generates utilization u_i for n tasks. The generation procedure is summarized as follows:

- For each task, utilization u_i is generated using *UUnifast-discard*;
- Period T_i is generated using a log-uniform distribution with a factor of 100 difference between the minimum and maximum possible task period. This presents a range of task periods from 10ms to 1s in real-time applications [DB11] [BDWZ12];
- C_i^{PE} is computed as $C_i^{PE} = u_i \cdot T_i$; and
- C_i^{EE} is computed as $C_i^{EE} = C_i^{PE} \cdot ce_i$, where ce_i is selected from a uniform random distribution in the range $[1.8, 2.3]$ which represents the variance of the execution time on different types of cores [Jef12].

Platforms

We have two types of cores (PE and EE) in the platforms and the core's power parameters are shown in Table 5.1 taken from [LSCS15]. In this experiment, we evaluate the effectiveness of our ASHM algorithm mainly on platforms with limited number of resources because on a platform with more resources our approach will always perform good, especially with more EE cores. Therefore, we conduct experiments on the following three limited platforms:

1. Platform 1: 2 PE cores and 2 EE cores
2. Platform 2: 2 PE cores and 3 EE cores
3. Platform 3: 3 PE cores and 2 EE cores

On the three platforms, we experiment with task sets with different U and a different number of tasks.

Comparison approaches

We compare our proposed ASHM algorithm with the following approaches in terms of energy consumption:

- FFD: Allocate E-tasks and NE-tasks to EE cores and PE cores, respectively, using FFD [CGJ97]. If E-tasks cannot be allocated to EE cores, then they are allocated to PE cores using FFD;
- WFD: Similar to FFD, but instead of FFD we use WFD [CGJ97] to allocate tasks;
- EFD: The allocation algorithm proposed in [ESAS14];
- m-pwr: The allocation algorithm proposed in [CKR14];

Comparison Metric

In the experimental results, we show the energy saving by using our ASHM compared to the above four reference approaches. The energy saving is computed as follows:

$$\text{Energy saving} = \frac{E_{ref} - E_{ASHM}}{E_{ref}} \cdot 100[\%] \tag{5.8}$$

where E_{ref} is the energy consumption of one of the four approaches given above and E_{ASHM} is the energy consumption of our proposed ASHM.

5.6.2 Experimental Results

All experimental results are plotted in Figure 5.1 to 5.6. For each point in the figures, we generate 100 random task sets and compute an average energy saving. Note that only when all reference approaches can schedule the generated task set we compute the energy saving using Equation (5.8). Our ASHM always can schedule more task sets than the other approaches because ASHM uses task-splitting. Since the schedulability advantage of the task-splitting approach has been reported in [BDWZ12], we do not compare the number of schedulable task sets in this work.

Impact of the Utilization

In this experiment, we fix the number of tasks for different platforms and then vary the total utilization to evaluate the effectiveness of ASHM. In order to have both NE-tasks and E-tasks in the generated task set, the number of tasks is fixed to 7 for all platforms. The results are plotted in Figure 5.1 to 5.3 where the y-axis is the energy saving computed using Equation (5.8) and the x-axis is the variable utilization. We can see that

our ASHM outperforms all allocation approaches in terms of energy efficiency. From the experimental results, we observe:

- The average energy saving by ASHM decreases as the total utilization increases. In the comparison between ASHM and WFD, EFD and m-pwr, this trend is easy to be observed although there is some variation due to the randomness of the generated task sets. The reason is that when we increase the total utilization, the slack space on the EE cores is reduced such that the task set cannot benefit from our ASHM too much. However, for FFD in Platform 1 and 3, see Figure 5.1 and 5.3, the energy saving increases until a point and then gradually decreases. The reason is that when we have task sets with a low utilization, FFD always tries to use the smallest number of cores to schedule tasks which might cause the PE cores to execute at a high frequency. The high frequency in turn leads to high energy consumption.

- ASHM saves more energy consumption on a platform with more EE cores, see Figure 5.2. The advantage of ASHM is to effectively utilize EE cores on the platform to achieve energy efficiency. More EE cores provide more space to split tasks and thus ASHM reduces more the energy consumption.

Impact of the Number of Tasks

In this experiment, we fix the utilization for different platforms and then vary the number of tasks to evaluate the effectiveness of ASHM. Since larger total utilization leads to smaller number of schedulable task sets, we fix the utilization to 2 for all platforms in order to compare our ASHM to the reference approaches on more schedulable task sets. We ensure that the number of tasks is greater than the number of cores, so we start with 4 tasks for Platform 1 and 5 tasks for Platform 2 and 3. The results are plotted in Figure 5.4 to 5.6.

Compared to the well-performed allocation approaches WFD and m-pwr, we can see that the energy saving is decreasing with the increasing number of tasks. The reason is that when the number of tasks increases with a fixed utilization, the tasks in the set become lighter, i.e., with a smaller utilization. Therefore, these tasks are easy to be allocated among the cores and then EE cores might be completely fulfilled or just have a little space for splitting of tasks. Therefore, ASHM cannot save too much in this case. However, as can be seen in Figure 5.4 and 5.6, compared to FFD, the energy saving by ASHM increases gradually. Since we have more tasks with a low utilization, FFD might allocate all tasks onto one core which will execute at a high frequency. However, since the dynamic power consumption still dominates the total power consumption, executing on two PE cores with lower frequencies is more energy-efficient than on one PE core with a high frequency.

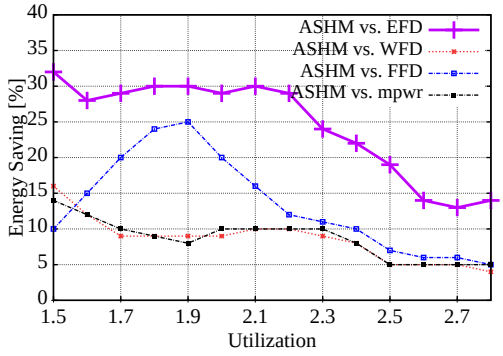

Figure 5.1: Varying U on platform with 2 PE cores and 2 EE cores

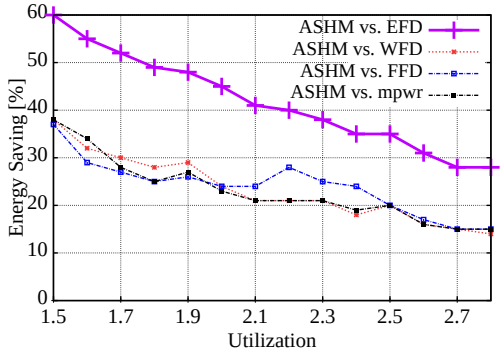

Figure 5.2: Varying U on platform with 2 PE cores and 3 EE cores

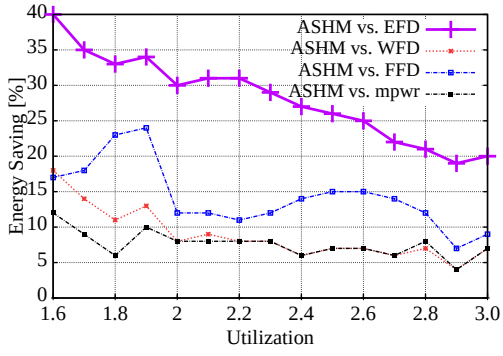

Figure 5.3: Varying U on platform with 3 PE cores and 2 EE cores

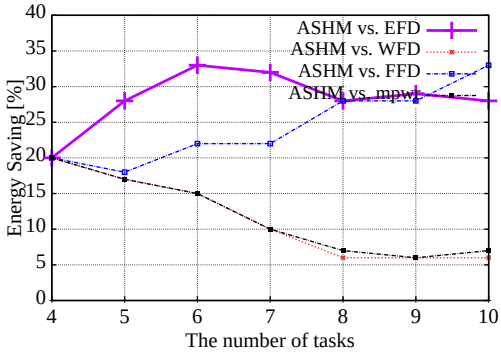

Figure 5.4: Varying the number of tasks on platform with 2 PE cores and 2 EE cores

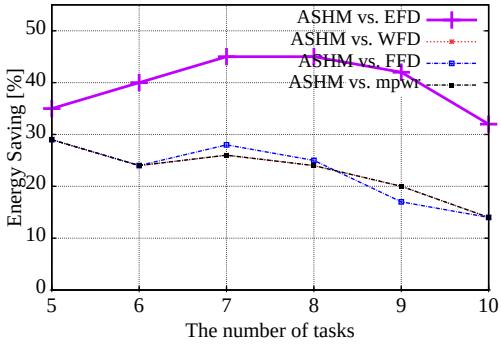

Figure 5.5: Varying the number of tasks on platform with 2 PE cores and 3 EE cores

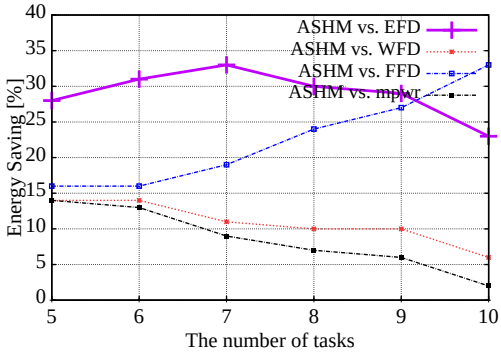

Figure 5.6: Varying the number of tasks on platform with 3 PE cores and 2 EE cores

5.7 Discussion

ASHM shows it effectiveness on per-core VFS systems via the experimental results in Section 5.6. We need to notice that a slight modification is capable of adapting ASHM to cluster heterogeneous multicore systems as considered in Chapter 4, but we leave this for the future work. On the other hand, since the *HRT* scheduling framework see in Section 2.3 can convert CSDF graphs into periodic task sets which can be fed as the input of the proposed ASHM, ASHM can also be applied by on a CSDF graph under the *HRT* scheduling.

Chapter 6

Schedulability Analysis of Imprecise Mixed-Criticality Systems

Di Liu, Jelena Spasic, Nan Guan, Gang Chen, Songran Liu, Todor Stefanov, Wang Yi,
"EDF-VD Scheduling of Mixed-Criticality Systems with Degraded Quality Guarantees",
"IEEE International Real-Time Systems Symposium (RTSS'16)", Porto, Portugal, Nov. 29 - Dec. 02, 2016.

As explained in Section 1.1.3, real-time applications with different criticality levels are being implemented on a shared computing platform in order to reduce Size, Weight, and Power (SWaP). We refer to this kind of integrated systems as Mixed-Criticality (MC) systems.

One of the core issues of MC systems stems from the certification authorities (CAs), as explained in Section 1.2 (Problem 3). Vestal in [Ves07] proposes a new model to specify real-time applications in MC systems. The new model captures the core features of MC systems and has received considerable attention since 2007. However, this classical MC model also receives some criticism from system designers [BB13], who complain that the model is too pessimistic in dropping off all low-criticality application tasks when any high-criticality application task overruns. Such an approach seriously disturbs the service of low-criticality tasks and influences the effectiveness of the whole system [BB13][SZ13].

To cope with the criticism and concerns from system designers, Burns and Baruah in [BB13] improve the classical MC model by introducing reduced WCETs for low-criticality tasks. Then, if any high-criticality task overruns its high-criticality WCET, instead of discarding low-criticality tasks, the improved MC model schedules low-criticality tasks with their reduced WCETs. Since the idea of reducing execution bud-

gets to keep tasks running is conceptually similar to the *imprecise computation model* [LLS+91][LSL+94], such MC systems we call *imprecise mixed criticality* (IMC) systems in [LSG+16].

Even though the IMC model is deemed to be a generalization of the classical MC model, it has not received sufficient attention. Only two works investigate the scheduling analysis of the IMC model. In [BB13], Burns and Baruah consider preemptive fixed-priority scheduling for the IMC model and extend the adaptive mixed criticality (AMC) [BBD11] approach to provide a schedulability test for the IMC model. Recently, Barauh *et al* in [BBG16] study the schedulability of the IMC model under Mixed-Criticality fluid scheduling (MC-fluid) [LPG+14].[1] Another widely-studied MC scheduling algorithm, EDF-VD [BBD+12], which has shown strong competence by both theoretical and empirical evaluations on the classical MC model [BBD+12, EY14, Eas13], has not been investigated for the IMC model. Therefore, in this chapter, we analyze the scheduability of the IMC model under EDF-VD scheduling. The novel technical contributions of our work include

- We propose a sufficient test for the IMC model under EDF-VD, - see Theorem 6.3.3 in Section 6.3;
- For the IMC model under EDF-VD, we derive a speedup factor function with respect to the utilization ratios of high criticality tasks and low criticality tasks - see Theorem 6.4.1 in Section 6.4. The derived speedup factor function enables us to quantify the suboptimality of EDF-VD and evaluate the impact of the utilization ratios on the speedup factor. We also compute the maximum value $4/3$ of the speedup factor function, which is equal to the speedup factor bound for the classical MC model [BBD+12].
- With extensive experiments, we show that for the IMC model, by using our proposed sufficient test, in most cases EDF-VD outperforms AMC [BB13] in terms of the number of schedulable task sets. Moreover, the experimental results validate the observations we have obtained for the speedup factor.

6.1 Related Work

Burns and Davis in [BD15] give a comprehensive review of the work on real-time scheduling for MC systems. Many of these works, e.g., [BBD+12] [EY14][Eas13], consider the classical MC model in which all low criticality tasks are discarded if the system switches to the high-criticality mode. In [BB13], Burns and Baruah discuss three approaches to keep some low criticality tasks running in *high*-criticality

[1] This work got public after our RTSS paper [LSG+16] was accepted.

mode. The first approach is to change the priority of low criticality tasks. However, for fixed-priority scheduling, de-prioritizing low criticality tasks cannot guarantee the execution of the low criticality tasks with a short deadline after the mode switches [BB13]. Similarly, for EDF, lowering the priority of low criticality tasks leads to a degraded service [HGST14]. In our work, we consider the IMC model which improves the schedulability of low criticality tasks in *high*-criticality mode by reducing their execution time. The IMC model can guarantee the regular service of a system by trading off the quality of the produced results. For some applications given in [LLS+91][LSL+94][RKKK14b], such trade-off is preferred.

The second approach in [BB13] is to extend the periods of low criticality tasks when the system mode changes to *high*-criticality mode such that the low criticality tasks execute less frequently to ensure their schedulability. Su *et al.* [SZ13][SGZ14] and Jan *et al.* [J+13] both consider this model. However, some applications might prefer an on-time result with a degraded quality rather than a delayed result with a perfect quality. Some example applications can be seen in [CLL90][LLS+91][LSL+94]. Then, the approach of extending periods is less useful for this kind of applications.

The last approach proposed in [BB13] is to reduce the execution budget of low criticality tasks when the system mode switches, i.e., the use of the IMC model studied in this chapter. In [BB13], the authors extend the AMC [BBD11] approach to test the schedulability of an IMC task set under fixed-priority scheduling. Recently, MC-fluid (MCF) scheduling of the IMC model is studied in [BBG16]. In practice, the MCF scheduling suffers from extremely high context switch overhead due to its very fine-grained scheduling units and thus it is difficult to be implemented on a real platform, whereas the EDF-VD scheduling considered in this chapter that is devised based on the the original EDF algorithm does not introduce too much scheduling overhead, thereby allowing to be implemented on a real platform. However, the schedulability problem for an IMC task set under EDF-VD [BBD+12], has not yet been addressed. Therefore, in our work, we study the schedulability of the IMC task model under EDF-VD and propose a sufficient test for it.

6.2 Preliminaries

This section first introduces the IMC task model and its execution semantics. Then, we give a brief explanation to the EDF-VD scheduling [BBD+12] and an example to illustrate the execution semantics of the IMC model under the EDF-VD scheduling.

6.2.1 Imprecise Mixed-Criticality Task Model

We use the *implicit-deadline sporadic* task model given in [BB13] where a task set Γ includes n tasks which are scheduled on a uniprocessor. Without loss of generality,

all tasks in Γ are assumed to start at time 0. Each task τ_i in Γ generates an infinite sequence of jobs $\{J_i^1, J_i^2 ...\}$ and is characterized by $\tau_i = \{T_i, D_i, L_i, \mathcal{C}_i\}$:

- T_i is the period or the minimal separation interval between two consecutive jobs;
- D_i denotes the relative task deadline, where $D_i = T_i$;
- $L_i \in \{LO, HI\}$ denotes the criticality (*low or high*) of a task. In this work, like in many previous research works [SZ13][HGST14][BBD$^+$12] [EY14][Eas13], we consider a dual-criticality MC model. Then, we split tasks into two task sets, $\Gamma_{LO} = \{\tau_i | L_i = LO\}$ and $\Gamma_{HI} = \{\tau_i | L_i = HI\}$;
- $\mathcal{C}_i = \{C_i^{LO}, C_i^{HI}\}$ is a list of WCETs, where C_i^{LO} and C_i^{HI} represent the WCET in *low*-criticality mode and the WCET in *high*-criticality mode, respectively. For a *high*-criticality task, it has $C_i^{LO} \leq C_i^{HI}$, whereas $C_i^{LO} \geq C_i^{HI}$ *for a low-criticality task, i.e., low-criticality task τ_i has a reduced WCET in high-criticality mode*.

Then each job J_i is characterized by $J_i = \{a_i, d_i, L_i, \mathcal{C}_i\}$, where a_i is the absolute release time and d_i is the absolute deadline. Note that if *low*-criticality task τ_i has $C_i^{HI} = 0$, it will be immediately discarded at the time of the switch to *high*-criticality mode. In this case, the IMC model behaves like the classical MC model.

The utilization of a task is used to denote the ratio between its WCET and its period. We define the following utilizations for an IMC task set Γ:

- For every task τ_i, it has $u_i^{LO} = \frac{C_i^{LO}}{T_i}$, $u_i^{HI} = \frac{C_i^{HI}}{T_i}$;
- For all *low*-criticality tasks, we have total utilizations

$$U_{LO}^{LO} = \sum_{\forall \tau_i \in \Gamma_{LO}} u_i^{LO}, \quad U_{LO}^{HI} = \sum_{\forall \tau_i \in \Gamma_{LO}} u_i^{HI}$$

- For all *high*-criticality tasks, we have total utilizations

$$U_{HI}^{LO} = \sum_{\forall \tau_i \in \Gamma_{HI}} u_i^{LO}, \quad U_{HI}^{HI} = \sum_{\forall \tau_i \in \Gamma_{HI}} u_i^{HI}$$

- For an IMC task set, we have

$$U^{LO} = U_{LO}^{LO} + U_{HI}^{LO}, \quad U^{HI} = U_{LO}^{HI} + U_{HI}^{HI}$$

6.2.2 Execution Semantics of the IMC Model

The execution semantics of the IMC model are similar to those of the classical MC model. The **major difference** occurs after a system switches to *high*-criticality mode. *Instead of discarding all low-criticality tasks, as it is done in the classical MC model, the IMC model tries to schedule low-criticality tasks with their reduced execution times* C_i^{HI}. The execution semantics of the IMC model are summarized as follows:

- The system starts in *low*-criticality mode, and remains in this mode as long as no *high*-criticality job overruns its *low*-criticality WCET C_i^{LO}. If any job of a *low*-criticality task tries to execute beyond its C_i^{LO}, the system will suspend it and launch a new job at the next period;
- If any job of *high*-criticality task executes for its C_i^{LO} time units without signaling completion, the system immediately switches to *high*-criticality mode;
- As the system switches to *high*-criticality mode, *if jobs of low-criticality tasks have completed execution for more than their* C_i^{HI} *but less than their* C_i^{LO}*, the jobs will be suspended till the tasks release new jobs for the next period. However, if jobs of low-criticality tasks have not completed their* C_i^{HI} *(*$\leq C_i^{LO}$*) by the switch time instant, the jobs will complete the left execution to* C_i^{HI} *after the switch time instant and before their deadlines. Hereafter, all jobs are scheduled using* C_i^{HI}. For *high*-criticality tasks, if their jobs have not completed their C_i^{LO} ($\leq C_i^{HI}$) by the switch time instant, all jobs will continue to be scheduled to complete C_i^{HI}. After that, all jobs are scheduled using C_i^{HI}.

Santy *et al.* [SGTG12] have shown that the system can switch back from the *high*-criticality mode to the *low*-criticality mode when there is an idle period and no *high*-criticality job awaits for execution. For the IMC model, we can use the same scenario to trigger the switch-back. In this work, we focus on the switch from *low*-criticality mode to *high*-criticality mode.

6.2.3 EDF-VD Scheduling

The challenge to schedule MC tasks with the EDF scheduling algorithm [LL73] is to deal with the overrun of *high*-criticality tasks when the system switches from *low*-criticality mode to *high*-criticality mode. Baruah *et al.* proposed in [BBD+12] to artificially tighten deadlines of jobs of *high*-criticality tasks in *low*-criticality mode such that the system can preserve execution budgets for the *high*-criticality tasks across mode switches. This approach is called *EDF with virtual deadlines* (EDF-VD).

Task	L	C_i^{LO}	C_i^{HI}	T_i	$\hat{D}_i$
τ_1	LO	3	2	9	
τ_2	HI	4	8	10	7

Table 6.1: Illustrative example

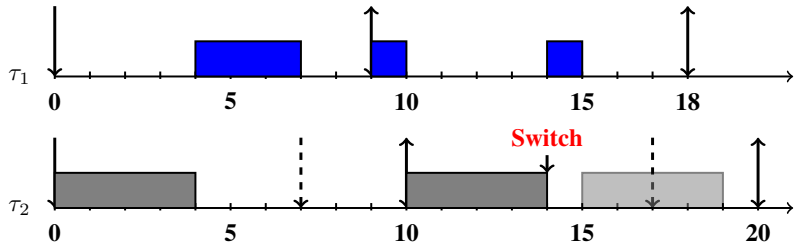

Figure 6.1: Scheduling of Example 6.1

6.2.4 An Illustrative Example

Here, we give a simple example to illustrate the execution semantics of the IMC model under EDF-VD. Table 6.1 gives two tasks, one *low*-criticality task τ_1 and one *high*-criticality task τ_2, where $\hat{D}_i$ is the virtual deadline. Figure 6.1 depicts the scheduling of the given IMC task set, where we assume that the mode switch occurs in the second period of τ_2. When the system switches to *high*-criticality mode, τ_2 will be scheduled by its original deadline 10 instead of its virtual deadline 7. Hence, τ_1 preempts τ_2 at the switch time instant. Since in *high*-criticality mode τ_1 only has execution budget of 2 , i.e., C_1^{HI}, τ_1 executes one unit and suspends. Then, τ_2 completes its left execution 4 ($C_2^{HI} - C_2^{LO}$) before its deadline.

6.3 Schedulability Analysis

In this section, we analyze the scheduability of the IMC model under EDF-VD scheduling and propose the first sufficient scheduability test. To ensure the timing correctness of the IMC model, we need to guarantee the scheduability for both *high*-criticality and *low*-criticality modes. Following, we demonstrate our analysis procedure and the formal theoretical proof.

6.3.1 Low Criticality Mode

We first ensure the schedulability of tasks when they are in *low*-criticality mode. As the task model is in *low*-criticality mode, the tasks can be considered as traditional

real-time tasks scheduled by EDF algorithm with virtual deadlines (VD). The following theorem is given in [BBD+12] for tasks scheduled in *low*-criticality mode.

Theorem 6.3.1 (Theorem 1 from [BBD+12])**.** *The following condition is sufficient for ensuring that EDF-VD successfully schedules all tasks in low-criticality mode:*

$$1 \geq \frac{U_{HI}^{LO}}{x} + U_{LO}^{LO} \tag{6.1}$$

where $x \in (0, 1)$ is used to uniformly modify the relative deadline of high-criticality tasks.

Since the IMC model behaves as the classical MC model in *low*-criticality mode, Theorem 6.3.1 holds for the IMC model as well.

6.3.2 High Criticality Mode

For *high*-criticality mode, the classical MC model discards all *low*-criticality jobs after the switch to *high*-criticality mode. In contrast, the IMC model keeps *low*-criticality jobs running but with degraded quality, i.e., a shorter execution time. So the schedulability condition in [BBD+12] does not work for the IMC model in the *high*-criticality mode. Thus, we need a new test for the IMC model in *high*-criticality mode.

To derive the sufficient test in *high*-criticality mode, suppose that there is a time interval $[0, t_2]$, where a first deadline miss occurs at t_2 and t_1 denotes the time instant of the switch to *high*-criticality mode in the time interval, where $t_1 < t_2$. Assume that $\mathcal{J}$ is the minimal set of jobs generated from task set Γ which leads to the first deadline miss at t_2. The minimality of $\mathcal{J}$ means that removing any job in $\mathcal{J}$ guarantees the schedulability of the rest of $\mathcal{J}$. Here, we introduce some notations for our later interpretation. Let variable η_i denote the cumulative execution time of task τ_i in the interval $[0, t_2]$. J_1 denotes a special *high*-criticality job which has switch time instant t_1 within its period (a_1, d_1), i.e, $a_1 < t_1 < d_1$. Furthermore, J_1 is the job with the earliest release time amongst all *high*-criticality jobs in $\mathcal{J}$ which execute in $[t_1, t_2)$. Moreover, we define a special type of job for *low*-criticality tasks which is useful for our later proofs.

Definition 6.3.1. A job J_i from *low*-criticality task τ_i is a carry-over job, if its absolute release time a_i is before and its absolute deadline d_i is after the switch time instant, i.e., $a_i < t_1 < d_i$.

With the notations introduced above, we have the following propositions,

Proposition 4 (Fact 1 from [BBD+12])**.** *All jobs in $\mathcal{J}$ that execute in $[t_1, t_2)$ have deadline $\leq t_2$.*

It is easy to observe that only jobs which have deadlines $\leq t_2$ are possible to cause a deadline miss at t_2. If a job has its deadline $> t_2$ and is still in set $\mathcal{J}$, it will contradict the minimality of $\mathcal{J}$.

Proposition 5. *The switch time instant t_1 has*

$$t_1 < (a_1 + x(t_2 - a_1)) \tag{6.2}$$

Proof. Let us consider a time instant $(a_1+x(d_1-a_1))$ which is the virtual deadline of job J_1. Since J_1 executes in time interval $[t_1, t_2)$, its virtual deadline $(a_1+x(d_1-a_1))$ must be greater than the switch time instant t_1. Otherwise, it should have completed its *low*-criticality execution before t_1, and this contradicts that it executes in $[t_1, t_2)$. Thus, it has

$$\begin{aligned} & t_1 < (a_1 + x(d_1 - a_1)) \\ \Rightarrow & t_1 < (a_1 + x(t_2 - a_1)) \quad (\text{since } d_1 \leq t_2) \end{aligned}$$

□

Proposition 6. *If a carry-over job J_i has its cumulative execution equal to $(d_i - a_i)u_i^{LO}$ and $u_i^{LO} > u_i^{HI}$, its deadline d_i is $\leq (a_1 + x(t_2 - a_1))$.*

Proof. For a carry-over job J_i, if it has its cumulative execution equal to $(d_i-a_i)u_i^{LO}$ and $u_i^{LO} > u_i^{HI}$, it should complete its C_i^{LO} execution before t_1. Otherwise, if job J_i has executed time units $C_i \in [C_i^{HI}, C_i^{LO})$ at time instant t_1, it will be suspended and will not execute after t_1.

Now, we will show that when job J_i completes its C_i^{LO} execution, its deadline is $d_i \leq (a_1 + x(t_2 - a_1))$. We prove this by contradiction. First, we suppose that J_i has its deadline $d_i > (a_1 + x(t_2 - a_1))$ and release time a_i. As shown above, job J_i completes its C_i^{LO} execution before t_1. Let us assume a time instant t^* as the latest time instant at which this carry-over job J_i starts to execute before t_1. This means that at this time instant all jobs in $\mathcal{J}$ with deadline $\leq (a_1 + x(t_2 - a_1))$ have finished their executions. This indicates that these jobs will not have any execution within interval $[t^*, t_2]$. Therefore, jobs in $\mathcal{J}$ with release time at or after time instant t^* can form a smaller job set which causes a deadline miss at t_2. Then, it contradicts the minimality of $\mathcal{J}$. Thus, carry-over job J_i with its cumulative execution time equal to $(d_i-a_i)u_i^{LO}$ and $u_i^{LO} > u_i^{HI}$ has its deadline $d_i \leq (a_1 + x(t_2 - a_1))$. □

With the propositions and notations given above, we derive an upper bound of the cumulative execution time η_i of *low*-criticality task τ_i.

Lemma 6.3.1. *For any low-criticality task τ_i, it has*

$$\eta_i \leq (a_1 + x(t_2 - a_1))u_i^{LO} + (1 - x)(t_2 - a_1)u_i^{HI} \tag{6.3}$$

Proof. If $u_i^{LO} = u_i^{HI}$, it is trivial to see that Lemma 6.3.1 holds. Below we focus on the case when $u_i^{LO} > u_i^{HI}$. If a system switches to *high*-criticality mode at t_1, then we know that *low*-criticality tasks are scheduled using C_i^{LO} before t_1 and using C_i^{HI} after t_1. To prove this lemma, we need to consider two cases, where τ_i releases a job within interval $(a_1, t_2]$ or it does not. We prove the two cases separately.

Case A (task τ_i releases a job within interval $(a_1, t_2]$): There are two sub-cases to be considered.

- **Sub-case 1 (No carry-over job):** The deadline of a job of *low*-criticality task τ_i coincides with switch time instant t_1. The cumulative execution time of *low*-criticality task τ_i within time interval $[0, t_2]$ can be bounded as follows,

$$\eta_i \leq (t_1 - 0) \cdot u_i^{LO} + (t_2 - t_1) \cdot u_i^{HI}$$

  Since $t_1 < (a_1 + x(t_2 - a_1))$ according to Proposition 5 and for *low*-criticality task τ_i it has $u_i^{LO} > u_i^{HI}$, then

$$\begin{aligned} &\eta_i < \big(a_1 + x(t_2 - a_1)\big)u_i^{LO} + \big(t_2 - \big(a_1 + x(t_2 - a_1)\big)\big)u_i^{HI} \\ \Leftrightarrow &\eta_i < (a_1 + x(t_2 - a_1))u_i^{LO} + (1 - x)(t_2 - a_1)u_i^{HI} \end{aligned}$$

- **Sub-case 2 (with carry-over job):** In this case, before the carry-over job, jobs of τ_i are scheduled with its C_i^{LO}. After the carry-over job, jobs of τ_i are scheduled with its C_i^{HI}. It is trivial to observe that for a carry-over job its maximum cumulative execution time can be obtained when it completes its C_i^{LO} within its period $[a_i, d_i]$, i.e., $(d_i - a_i)u_i^{LO}$. Considering the maximum cumulative execution for the carry-over job, we then have for *low*-criticality task τ_i,

$$\begin{aligned} &\eta_i \leq (a_i - 0)u_i^{LO} + (d_i - a_i)u_i^{LO} + (t_2 - d_i)u_i^{HI} \\ \Leftrightarrow &\eta_i \leq d_i u_i^{LO} + (t_2 - d_i)u_i^{HI} \end{aligned}$$

  Proposition 6 shows that as J_i has its cumulative execution equal to $(d_i - a_i) \cdot u_i^{LO}$, it has $d_i \leq (a_1 + x(t_2 - a_1))$. Given that $u_i^{LO} > u_i^{HI}$ for *low*-criticality task, we have

$$\begin{aligned} &\eta_i \leq d_i u_i^{LO} + (t_2 - d_i)u_i^{HI} \\ \Rightarrow &\eta_i \leq \big(a_1 + x(t_2 - a_1)\big)u_i^{LO} + \big(t_2 - \big(a_1 + x(t_2 - a_1)\big)\big)u_i^{HI} \\ \Leftrightarrow &\eta_i \leq (a_1 + x(t_2 - a_1))u_i^{LO} + (1 - x)(t_2 - a_1)u_i^{HI} \end{aligned}$$

Case B (task τ_i does not release a job within interval $(a_1, t_2]$): In this case, let J_i denote the last release job of task τ_i before a_1 and a_i and d_i are its absolute release time and absolute deadline, respectively. If $d_i \leq t_1$, we have

$$\eta_i = (a_i - 0)u_i^{LO} + (d_i - a_i) \cdot u_i^{LO} = d_i u_i^{LO}$$

If $d_i > t_1$, J_i is a carry-over job. As we discussed above, the maximum cumulative execution time of carry-over job J_i is $(d_i - a_i)u_i^{LO}$, so we have

$$\eta_i \leq (a_i - 0)u_i^{LO} + (d_i - a_i) \cdot u_i^{LO} \Leftrightarrow \eta_i \leq d_i u_i^{LO}$$

Similarly, according to Proposition 6, we obtain,

$$\begin{aligned} &\eta_i \leq d_i \cdot u_i^{LO} \leq (a_1 + x(t_2 - a_1))u_i^{LO} \\ \Rightarrow &\eta_i < (a_1 + x(t_2 - a_1))u_i^{LO} + \big(t_2 - \big(a_1 + x(t_2 - a_1)\big)\big)u_i^{HI} \\ \Leftrightarrow &\eta_i < (a_1 + x(t_2 - a_1))u_i^{LO} + (1 - x)(t_2 - a_1)u_i^{HI} \end{aligned}$$

□

Lemma 6.3.1 gives the upper bound of the cumulative execution time of a *low*-criticality task in *high*-criticality mode. In order to derive the sufficient test for the IMC model in *high*-criticality mode, we need to upper bound the cumulative execution time of *high*-criticality tasks.

Proposition 7 (Fact 3 from [BBD+12])**.** *For any high-criticality task τ_i, it holds that*

$$\eta_i \leq \frac{a_1}{x}u_i^{LO} + (t_2 - a_1)u_i^{HI} \tag{6.4}$$

Proposition 7 is used to bound the cumulative execution of the *high*-criticality tasks. Since in the IMC model the *high*-criticality tasks are scheduled as in the classical MC model, Proposition 7 holds for the IMC model as well. With Lemma 6.3.1 and Proposition 7, we can derive the sufficient test for the IMC model in *high*-criticality mode.

Theorem 6.3.2. *The following condition is sufficient for ensuring that EDF-VD successfully schedules all tasks in high-criticality mode:*

$$xU_{LO}^{LO} + (1 - x)U_{LO}^{HI} + U_{HI}^{HI} \leq 1 \tag{6.5}$$

Proof. Let N denote the cumulative execution time of all tasks in $\Gamma = \Gamma_{LO} \cup \Gamma_{HI}$ over interval $[0, t_2]$. We have

$$N = \sum_{\forall \tau_i \in \Gamma_{LO}} \eta_i + \sum_{\forall \tau_i \in \Gamma_{HI}} \eta_i$$

By using Lemma 6.3.1 and Proposition 7, N is bounded as follows

$$\begin{aligned}
N \leq & \sum_{\forall \tau_i \in \Gamma_{LO}} \left(\big(a_1 + x(t_2 - a_1)\big) u_i^{LO} + (1-x)(t_2 - a_1) u_i^{HI} \right) \\
& + \sum_{\forall \tau_i \in \Gamma_{HI}} \left(\frac{a_1}{x} u_i^{LO} + (t_2 - a_1) u_i^{HI} \right) \\
\Leftrightarrow N \leq & (a_1 + x(t_2 - a_1)) U_{LO}^{LO} + (1-x)(t_2 - a_1) U_{LO}^{HI} \\
& + \frac{a_1}{x} U_{HI}^{LO} + (t_2 - a_1) U_{HI}^{HI} \\
\Leftrightarrow N \leq & a_1 (U_{LO}^{LO} + \frac{U_{HI}^{LO}}{x}) + x(t_2 - a_1) U_{LO}^{LO} \\
& + (1-x)(t_2 - a_1) U_{LO}^{HI} + (t_2 - a_1) U_{HI}^{HI}
\end{aligned} \tag{6.6}$$

Since the tasks must be schedulable in *low*-criticality mode, the condition given in Theorem 6.3.1 holds and we have $1 \geq (U_{LO}^{LO} + \frac{U_{HI}^{LO}}{x})$. Hence,

$$\begin{aligned}
N \leq & a_1 + x(t_2 - a_1) U_{LO}^{LO} \\
& + (1-x)(t_2 - a_1) U_{LO}^{HI} + (t_2 - a_1) U_{HI}^{HI}
\end{aligned} \tag{6.7}$$

Since time instant t_2 is the first deadline miss, it means that there is no idle time instant within interval $[0, t_2]$. Note that if there is an idle instant, jobs from set $\mathcal{J}$ which have release time at or after the latest idle instant can form a smaller job set causing deadline miss at t_2 which contradicts the minimality of $\mathcal{J}$. Then, we obtain

$$\begin{aligned}
& N = \left(\sum_{\forall \tau_i \in \Gamma_{LO}} \eta_i + \sum_{\forall \tau_i \in \Gamma_{HI}} \eta_i \right) > t_2 \\
\Rightarrow & a_1 + x(t_2 - a_1) U_{LO}^{LO} + (1-x)(t_2 - a_1) U_{LO}^{HI} + (t_2 - a_1) U_{HI}^{HI} \\
& > t_2 \\
\Leftrightarrow & x(t_2 - a_1) U_{LO}^{LO} + (1-x)(t_2 - a_1) U_{LO}^{HI} + (t_2 - a_1) U_{HI}^{HI} \\
& > t_2 - a_1 \\
\Leftrightarrow & x U_{LO}^{LO} + (1-x) U_{LO}^{HI} + U_{HI}^{HI} > 1
\end{aligned}$$

By taking the contrapositive, we derive the sufficient test for the IMC model when it is in *high*-criticality mode:

$$x U_{LO}^{LO} + (1-x) U_{LO}^{HI} + U_{HI}^{HI} \leq 1$$

□

Note that if $U_{LO}^{HI} = 0$, i.e., no *low*-criticality tasks are scheduled after the system switches to *high*-criticality mode, our Theorem 6.3.2 is the same as the sufficient

test (Theorem 2 in [BBD+12]) for the classical MC model in *high*-criticality mode. Hence, our Theorem 6.3.2 actually is a generalized schedulability condition for (I)MC tasks under EDF-VD.

By combining Theorem 6.3.1 (see Section 6.3.1) and our Theorem 6.3.2, we prove the following theorem,

Theorem 6.3.3. *Given an IMC task set, if*

$$U_{HI}^{HI} + U_{LO}^{LO} \leq 1 \tag{6.8}$$

then the IMC task set is schedulable by EDF; otherwise, if

$$\frac{U_{HI}^{LO}}{1 - U_{LO}^{LO}} \leq \frac{1 - (U_{HI}^{HI} + U_{LO}^{HI})}{U_{LO}^{LO} - U_{LO}^{HI}} \tag{6.9}$$

where

$$U_{HI}^{HI} + U_{LO}^{HI} < 1 \text{ and } U_{LO}^{LO} < 1 \text{ and } U_{LO}^{LO} > U_{LO}^{HI} \tag{6.10}$$

then this IMC task set can be scheduled by EDF-VD with a deadline scaling factor x arbitrarily chosen in the following range

$$x \in \left[\frac{U_{HI}^{LO}}{1 - U_{LO}^{LO}}, \ \frac{1 - (U_{HI}^{HI} + U_{LO}^{HI})}{U_{LO}^{LO} - U_{LO}^{HI}} \right]$$

Proof. Total utilization $U \leq 1$ is the exact test for EDF on a uniprocessor system. If the condition in (6.8) is met, the given task set is *worst-case reservation* [BBD+12] schedulable under EDF, i.e., the task set can be scheduled by EDF without deadline scaling for *high*-criticality tasks and execution budget reduction for *low*-criticality tasks. Now, we prove the second condition given by (6.9). From Theorem 6.3.1, we have,

$$x \geq \frac{U_{HI}^{LO}}{1 - U_{LO}^{LO}}$$

From Theorem 6.3.2, we have

$$\begin{aligned} & xU_{LO}^{LO} + (1 - x)U_{LO}^{HI} + U_{HI}^{HI} \leq 1 \\ \Leftrightarrow & x \leq \frac{1 - (U_{HI}^{HI} + U_{LO}^{HI})}{U_{LO}^{LO} - U_{LO}^{HI}} \end{aligned}$$

Therefore, if $\frac{U_{HI}^{LO}}{1-U_{LO}^{LO}} \leq \frac{1-(U_{HI}^{HI}+U_{LO}^{HI})}{U_{LO}^{LO}-U_{LO}^{HI}}$, the schedulability conditions of both Theorem 6.3.1 and 6.3.2 are satisfied. Thus, the IMC tasks are schedulable under EDF-VD. □

6.4 Speedup Factor

The speedup factor bound is a useful metric to compare the worst-case performance of different MC scheduling algorithms. The following is the definition of the speedup factor for an MC scheduling algorithm.

Definition 6.4.1 (from [BBD+12])**.** The *speedup factor* of an algorithm $\mathcal{A}$ for scheduling MC systems is the smallest real number $f \geq 1$ such that any task system that is schedulable by a hypothetical optimal clairvoyant scheduling algorithm[2] on a unit-speed processor is correctly scheduled by algorithm $\mathcal{A}$ on a speed-f processor.

Informally speaking, by increasing the processor's speed, a non-optimal scheduling algorithm is able to schedule the task sets which are deemed to be unschedulable by the non-optimal scheduling algorithm but schedulable by an optimal scheduling algorithm on the processor without speed increase. The speedup factor actually computes how much the processor needs to speed up such that the non-optimal scheduling algorithm achieves the same scheduling performance as an optimal scheduling algorithm. The smaller speedup factor indicates a better scheduling performance for the non-optimal scheduling algorithm. The speedup factor bound for the classical MC model under EDF-VD [BBD+12] has been shown to be $4/3$.

In the following, we prove the speedup factor of the IMC model under EDF-VD scheduling. For notational simplicity, we define

$$U_{HI}^{HI} = c, \quad U_{HI}^{LO} = \alpha \times c$$
$$U_{LO}^{LO} = b, \quad U_{LO}^{HI} = \lambda \times b$$

where $\alpha \in (0, 1]$ and $\lambda \in [0, 1]$. α denotes the utilization ratio between U_{HI}^{LO} and U_{HI}^{HI}, while λ denotes the utilization ratio between U_{LO}^{HI} and U_{LO}^{LO}.

First, let us analyze the speedup factor of two corner cases. When $\alpha = 1$, i.e., $U_{HI}^{LO} = U_{HI}^{HI}$, this means that there is no mode-switch. Therefore, the task set is scheduled by the traditional EDF, i.e., the task set is schedulable if and only if $U_{LO}^{LO} + U_{HI}^{LO} \leq 1$. Since EDF is the optimal scheduling algorithm on a uniprocessor system, the speedup factor thus is 1. When $\lambda = 1$, i.e., $U_{LO}^{LO} = U_{LO}^{HI}$, if the task set is schedulable in *high*-criticality mode, it must hold $U_{HI}^{HI} + U_{LO}^{LO} \leq 1$ by Theorem 6.3.2. Then it is scheduled by the traditional EDF and thus the speedup factor is 1 as well.

In our work, instead of generating a single speedup factor bound, we derive a speedup factor function with respect to (α, λ). This speedup factor function enables

[2] A 'clairvoyant' scheduling algorithm knows all run-time information, e.g., when the mode switch will occur, prior to run-time.

us to quantify the suboptimality of EDF-VD for the IMC model in terms of speedup factor (by our proposed sufficient test) and to evaluate the impact of the utilization ratio on the schedulability of an IMC task set under EDF-VD.

First, we strive to find a minimum speed s ($\leq$1) for a clairvoyant optimal MC scheduling algorithm such that any implicit-deadline IMC task set which is schedulable by the clairvoyant optimal MC scheduling algorithm on a speed-s processor can satisfy the schedulability test given in Theorem 6.3.3, i.e., schedulable under EDF-VD on a unit-speed processor. Then, we can compute the speed-up factor by simply computing $1/s$.

Lemma 6.4.1. *Given* $b, c \in [0, 1]$, $\alpha \in (0, 1)$, $\lambda \in [0, 1)$, *and*

$$\max\{b + \alpha c, \lambda b + c\} \leq S(\alpha, \lambda) \tag{6.11}$$

where

$$S(\alpha, \lambda) = \frac{(1 - \alpha\lambda)((2 - \alpha\lambda - \alpha) + (\lambda - 1)\sqrt{4\alpha - 3\alpha^2})}{2(1 - \alpha)(\alpha\lambda - \alpha\lambda^2 - \alpha + 1)}$$

then it guarantees

$$\frac{\alpha c}{1 - b} \leq \frac{1 - (c + \lambda b)}{b - \lambda b} \tag{6.12}$$

Proof. Suppose that λ and α are constants and we have a real number $s \leq 1$, where $\max\{b + \alpha c, \lambda b + c\} \leq s$. We need to find the minimum of s which guarantees that *any* $b, c \in [0, 1]$ ensure (6.12). First, $\max\{b + \alpha c, \lambda b + c\} \leq s$ implies

$$b + \alpha c \leq s \tag{6.13}$$

$$\lambda b + c \leq s \tag{6.14}$$

Then, condition (6.12) can be written as follows,

$$\lambda b^2 + (\alpha\lambda - \alpha + 1)bc - (\lambda + 1)b - c + 1 \geq 0 \tag{6.15}$$

Inequalities (6.13)(6.14)(6.15) define a feasible space in the three-dimension space, respectively. In Figure 6.2, the space above the plane is a feasible space satisfying (6.13), where the plane corresponds to $b + \alpha c = s$. For (6.14), $\lambda b + c = s$ draws a plane and the feasible space is above the plane shown in Figure 6.3. Similarly, when (6.15) makes its right-hand-side equal to the left-hand-side, we draw a vertical curved surface seen in Figure 6.4 and the space inside the vertical surface is the feasible space (the opposite of the arrow direction). We need to find the *minimum* of s in the feasible space (above the two planes and inside the vertical surface) such that **any** b and c that meet (6.11) satisfy (6.12). Since $\max\{b + \alpha c, \lambda b + c\} = s$ is strictly increasing, to ensure that condition (6.12) hold for *any* b and c, we strive to minimize

$\max\{b+\alpha c, \lambda b+c\}$ in the feasible space. Then, this problem can be transformed into another form, where, instead of minimizing $\max\{b + \alpha c, \lambda b + c\}$ inside the vertical surface, we minimize the value of $\max\{b+\alpha c, \lambda b+c\}$ in the space outside the vertical surface[3] which is defined by

$$\lambda b^2 + (\alpha\lambda - \alpha + 1)bc - (\lambda + 1)b - c + 1 \leq 0 \tag{6.16}$$

This is equivalent to the minimization of s with the above constraint. Then, the minimization problem is formulated as follows,

$$\begin{aligned}
&\text{minimize} && s && (6.17)\\
&\text{subject to} && b + \alpha c \leq s && (6.18)\\
& && \lambda b + c \leq s && (6.19)\\
& && \lambda b^2 + (\alpha\lambda - \alpha + 1)bc - (\lambda + 1)b - c + 1 \leq 0 && (6.20)\\
& && 0 \leq b \leq 1, \quad 0 \leq c \leq 1 && (6.21)
\end{aligned}$$

where α and λ are constant and s, b, c are variables. If $S(\alpha, \lambda)$ is the optimal solution of the optimization problem (6.17), then Lemma 6.4.1 is proven.

Below, we prove that $S(\alpha, \lambda)$ is the optimal solution of the optimization problem (6.17)[4]

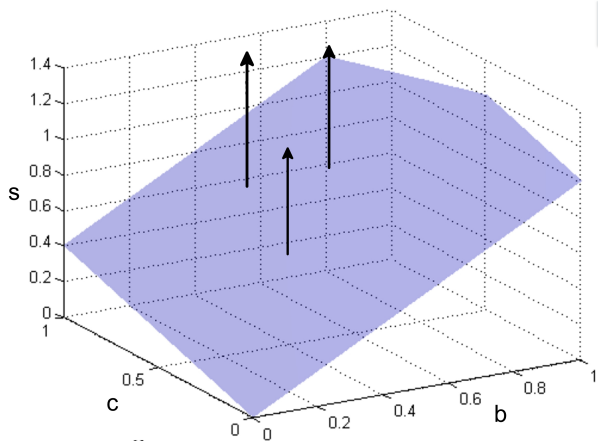

Figure 6.2: plane 1

[3] As the arrows direct

[4] This optimization problem is a non-convex problem and thus we cannot use general convex optimization techniques such as the Karush-Kuhn-Tucker (KKT) approach [KT51] to solve it.

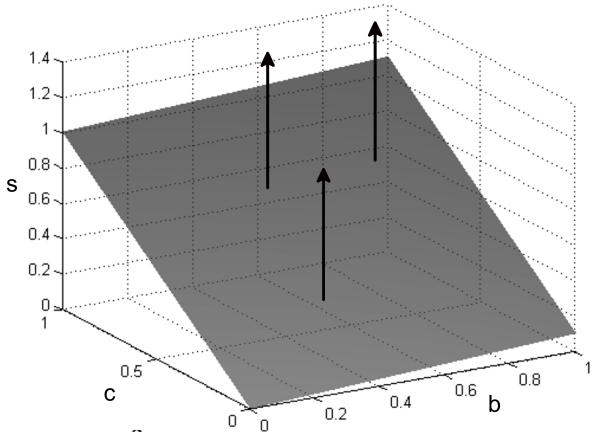

Figure 6.3: plane 2

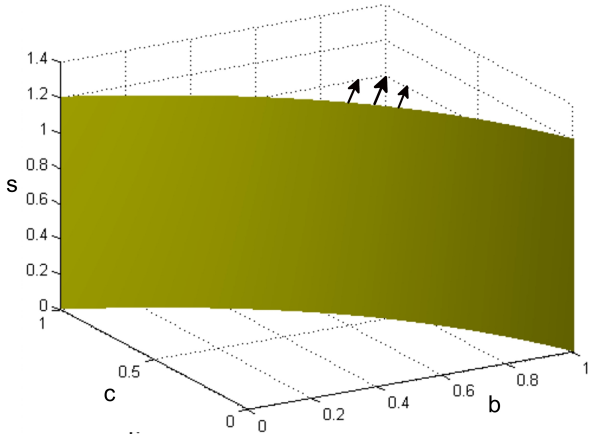

Figure 6.4: vertical surface

As stated before, the feasible solutions subject to these three constraints (6.18), (6.19) and (6.20) must be above both planes and outside the vertical curved surface. First assume that we have a point (b'_0, c'_0, s'_0) which satisfies all constraints but is not on the vertical surface. If we connect the origin $(0, 0, 0)$ and (b'_0, c'_0, s'_0), this line must have an intersection point (b^*_0, c^*_0, s^*_0) with the vertical surface. It is easy to observe that $s^*_0 < s'_0$ - see in Figure 6.5. This means that any point which is not on the vertical surface can find a point with smaller value of s on the vertical surface which satisfies all constraints. Therefore, the point with the minimum s must be on the vertical surface. Similarly, the minimum s must be on one of the two planes.

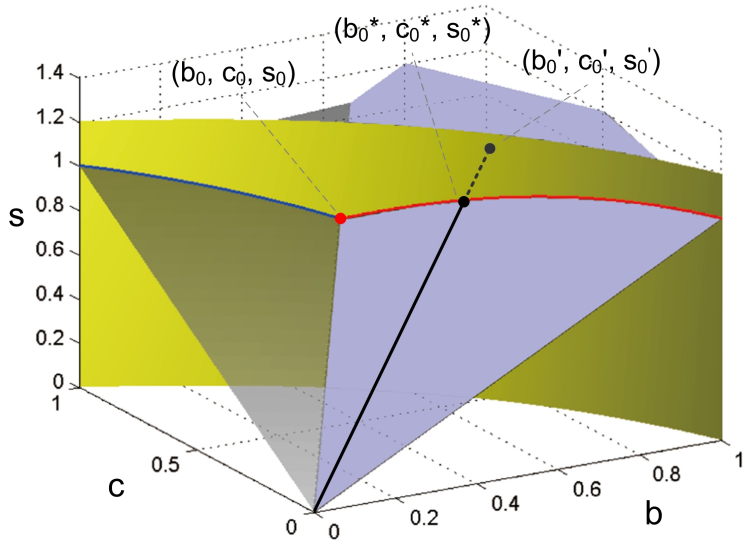

Figure 6.5: 3D space of optimization problem (6.17)

Otherwise, if it is not on any plane, we always can find a projected point on one plane which has a smaller value of s.

We have shown above that to obtain the minimum value of s the point must be on the vertical surface and one plane. Then, the two planes have an intersection line and this line intersects with the vertical surface at a point denoted by (b_0, c_0, s_0). By taking constraints (6.18)(6.19) and (6.20), we formulate a piece-wise function of s with respect to b as follows.

$$s(b) = \begin{cases} \frac{(\alpha\lambda^2-\alpha\lambda)b^2+b-1}{(\alpha\lambda-\alpha+1)b-1} & 0 < b \leq b_0 \\ \frac{(1-\alpha)b^2+(\alpha\lambda+\alpha-1)b-\alpha}{(\alpha\lambda-\alpha+1)b-1} & b_0 < b \leq 1 \end{cases} \tag{6.22}$$

This function covers all points which are on the vertical surface and one plane and at same time satisfy all constraints. By doing some calculus, we know that Equation (6.22) is monotonically decreasing in $(0, b_0]$ and monotonically increasing in $(b_0, 1]$. Therefore, the minimum value of Equation (6.22) can be obtained at (b_0, c_0, s_0). The complete proof is given by Lemma 1 in Appendix I. It means that we can obtain the optimal solution of problem (6.17) by solving the following system of equations.

$$\begin{cases} b_0 + \alpha c_0 = s_0 \\ \lambda b_0 + c_0 = s_0 \\ \lambda b_0^2+(\alpha\lambda-\alpha+1)b_0c_0-(\lambda+1)b_0-c_0+1 = 0 \end{cases} \tag{6.23}$$

By joining the first two equations we have $c_0 = \frac{1-\lambda}{1-\alpha} \times b_0$, and applying it to the last equation in (6.23) gives

$$(-\alpha\lambda^2 + \alpha\lambda - \alpha + 1)b_0^2 + (\alpha\lambda + \alpha - 2)b_0 + (1 - \alpha) = 0$$

By the well-known Quadratic Formula we get the two roots of the above quadratic equation.

$$b_0^1 = \frac{(2 - \alpha\lambda - \alpha) + (1 - \lambda)\sqrt{-3\alpha^2 + 4\alpha}}{2(-\alpha\lambda^2 + \alpha\lambda - \alpha + 1)} \tag{6.24}$$

$$b_0^2 = \frac{(2 - \alpha\lambda - \alpha) - (1 - \lambda)\sqrt{-3\alpha^2 + 4\alpha}}{2(-\alpha\lambda^2 + \alpha\lambda - \alpha + 1)} \tag{6.25}$$

We can prove that b_0^2 is larger than 1 and thus should be dropped (since we require $0 \leq b \leq 1$), while b_0^1 is in the range of $[0, 1]$. The detailed proof is given by Lemma 2 in Appendix I. As a result, we obtain the optimal solution $(b_0^1, \frac{1-\lambda}{1-\alpha}b_0^1, \frac{1-\alpha\lambda}{1-\alpha}b_0^1)$ for Equation (6.23). Thus, we have

$$\begin{aligned} S(\alpha, \lambda) &= \frac{1 - \alpha\lambda}{1 - \alpha} b_0^1 \\ &= \frac{(1 - \alpha\lambda)((2 - \alpha\lambda - \alpha) + (\lambda - 1)\sqrt{4\alpha - 3\alpha^2})}{2(1 - \alpha)(\alpha\lambda - \alpha\lambda^2 - \alpha + 1)} \end{aligned}$$

Therefore, Lemma 6.4.1 is proven. □

Lemma 6.4.1 shows that any IMC task set that is schedulable by an optimal clairvoyant MC scheduling algorithm on a speed-$S(\alpha, \lambda)$ is schedulable by EDF-VD on a unit-speed processor. Therefore, we can compute the speedup factor of EDF-VD by $1/S(\alpha, \lambda)$.

Theorem 6.4.1. *The speedup factor of EDF-VD with IMC task sets is*

$$f = \frac{2(1 - \alpha)(\alpha\lambda - \alpha\lambda^2 - \alpha + 1)}{(1 - \alpha\lambda)((2 - \alpha\lambda - \alpha) + (\lambda - 1)\sqrt{4\alpha - 3\alpha^2})}$$

Proof. Follow the explanation given above. □

The speedup factor is shown to be a function of α and λ. Figure 6.6 plots the 3D image of this function and Table 6.2 lists some of the values with different α and λ. By doing some calculus, we obtain the maximum value 1.333 $(4/3)$ of the speedup factor function when $\lambda = 0$ and $\alpha = \frac{1}{3}$, which is highlighted in Figure 6.6 and Table

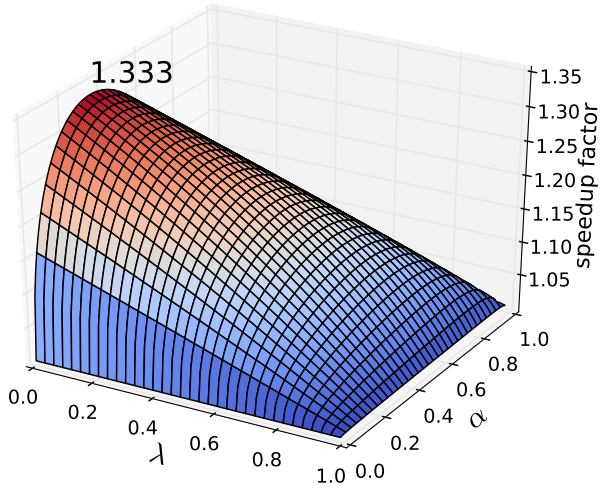

Figure 6.6: 3D image of the speedup factor w.r.t α and λ

λ \ α	0.1	0.3	1/3	0.5	0.7	0.9	1
0	1.254	1.332	**1.333**	1.309	1.227	1.091	1
0.1	1.231	1.308	1.310	1.293	1.219	1.090	1
0.3	1.183	1.256	1.259	1.254	1.201	1.087	1
0.5	1.134	1.195	1.200	1.206	1.174	1.083	1
0.7	1.082	1.126	1.130	1.143	1.133	1.074	1
0.9	1.028	1.046	1.048	1.056	1.061	1.048	1
1	1	1	1	1	1	1	1

Table 6.2: The speedup factor w.r.t α and λ

6.2. We see that the speedup factor bound is achieved when the task set is a classical MC task set. From Figure 6.6 and Table 6.2, we observe different trends for the speedup factor with respect to α and λ.

- First, given a fixed λ, the speedup factor is not a monotonic function with respect to α. The relation between α and the speedup factor draws a downward parabola. Therefore, a straightforward conclusion regarding the impact of α on the speedup factor cannot be drawn.

- Given a fixed α, the speedup factor is a monotonically decreasing function with respect to increasing λ. It is seen that increasing λ leads to a smaller value of the speedup factor. *This means that a larger λ brings a positive effect on the schedulability of an IMC task set*.

6.5 Experimental Evaluation

In this section, we conduct experiments to evaluate the effectiveness of the proposed sufficient test for the IMC model in terms of schedulable task sets (acceptance ratio). Moreover, we conduct experiments to verify the two observations stated at the end of Section 6.4 regarding the impact of α and λ on the average acceptance ratio. Our experiments are based on randomly generated MC tasks. We use a task generation approach, similar to that used in [Eas13][EY14], to randomly generate IMC task sets to evaluate the proposed sufficient test. Each task τ_i is generated based on the following procedure,

- pCriticality is the probability that the generated task is a *high*-criticality task; pCriticality$\in [0, 1]$.

- Period T_i is randomly selected from the range $[100, 1000]$.

- In order to have sufficient number of tasks in a task set, utilization u_i is randomly drawn from the range$[0.05, 0.2]$.

- For any task τ_i, $C_i^{LO} = u_i * T_i$.

- $R \geq 1$ denotes the ratio C_i^{HI}/C_i^{LO} for every *high*-criticality task. If $L_i = HI$, we set $C_i^{HI} = R * C_i^{LO}$. It is easy to see that α used in the speedup factor function is equal to $\frac{1}{R}$;

- $\lambda \in (0, 1]$ denotes the ratio C_i^{HI}/C_i^{LO} for every *low*-criticality task. If $L_i = LO$, we set $C_i^{HI} = \lambda * C_i^{LO}$.

In the experiment, we generate IMC task sets with different target utilization U. Each task set is generated as follows. Given a target utilization U, we first initialize an empty task set. Then, we generate task τ_i according to the task generation procedure introduced above and add the generated task to the task set. The task set generation stops as we have

$$U - 0.05 \leq U_{avg} \leq U + 0.05$$

where

$$U_{avg} = \frac{U^{LO} + U^{HI}}{2}$$

is the average total utilization of the generated task set. If adding a new task makes $U_{avg} > U + 0.05$, then the added task will be removed and a new task will be generated and added to the task set till the condition is met.

6.5.1 Comparison with AMC [BB13]

In the first experiment, we compare EDF-VD by using our proposed test to the AMC approach in [BB13] in terms of average acceptance ratio. In this experiment, R is randomly selected from a uniform distribution $[1.5, 2.5]$. With different λ and pCriticality settings, we vary U_{avg} from 0.4 to 0.95 with step of 0.05, to evaluate the effectiveness of the proposed sufficient test in terms of the average acceptance ratios. We generate 10,000 task sets for each given U_{avg}. Since all experimental results follow the similar trend, in this section, we only present the experimental results when pCriticality$= 0.5$. Results with different pCriticality settings can be found in Appendix III. The results are shown in Figure 6.7-6.9, where the x-axis denotes the varying U_{avg} and the y-axis denotes the acceptance ratio. In the figures, let EDF-VD and AMC denote our proposed schedulability test and the one proposed in [BB13], respectively. In most cases, EDF-VD outperforms AMC in terms of acceptance ratio. We observe the following trends:

1. When $U_{avg} \in [0.5, 0.8]$, EDF-VD always outperforms AMC in terms of acceptance ratio. However, if $U_{avg} > 0.8$ and $\lambda = 0.3$ or 0.5, AMC performs better than EDF-VD. The same trend is also found for the classical MC model under EDF-VD and AMC, see in [EY14].

2. By comparing figures in Figure 6.7-6.9, we see that the average acceptance ratio improves when λ increases. This confirms the observation for the speedup factor we stated at the end of Section 6.4. The increasing λ leads to a smaller speedup factor. As a result, it provides a better schedulability. We need to notice that when λ increases, not only EDF-VD improves its acceptance ratio but the acceptance ratio of AMC [BB13] also improves.

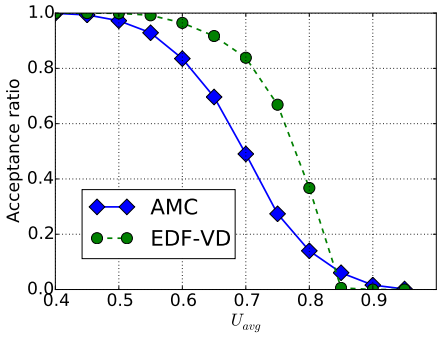

Figure 6.7: $\lambda = 0.3$

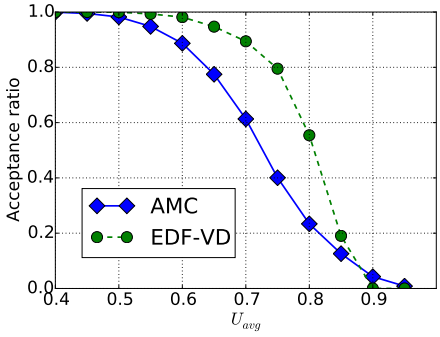

Figure 6.8: $\lambda = 0.5$

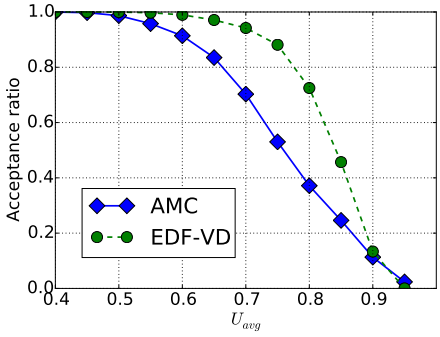

Figure 6.9: $\lambda = 0.7$

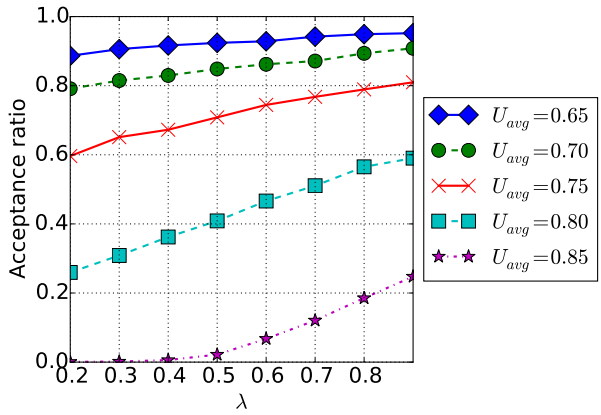

Figure 6.10: Impact of λ

6.5.2 Impact of α and λ

In the first experiment, we compare our proposed sufficient test to the existing AMC approach. In this section, we conduct experiments to further evaluate the impact of λ and α $(1/R)$ on the acceptance ratio. In this experiment, we select $U_{avg} = \{0.65, 0.7, 0.75, 0.8, 0.85\}$ to conduct experiments. We fix U_{avg} to a certain utilization and vary λ and α to evaluate the impact.

We first show the results for λ. The results are depicted in Figure 6.10, where the x-axis denotes the value of λ from 0.2 to 0.9 with step of 0.1 and the y-axis denotes the average acceptance ratio. R is randomly selected from a uniform distribution $[1.5, 2.5]$ and pCriticality$= 0.5$. Similarly, 10,000 task sets are generated for each point in the figures. A clear trend can be observed that the acceptance ratio increases as λ increases. This trend confirms the positive impact of increasing λ on the schedulability which we have observed in Section 6.4.

Next we conduct experiments to evaluate the impact of α on the schedulability. Similarly, we fix U_{avg} and vary α to carry out the experiments. Due to $\alpha = \frac{1}{R}$, if α is given, we compute the corresponding R to generate task sets. The results are depicted in Figure 6.11, where $\lambda = 0.5$. The x-axis denotes the varying α from 0.1 to 0.9 with step of 0.1. while the y-axis denotes the average acceptance ratio. First, from Table 6.2, we see that with increasing α the speedup factor first increases till a point. This means within this range the scheduling performance of EDF-VD gradually decreases. After that point, the speedup factor decreases which means the scheduling performance of EDF-VD gradually improves. The experimental results confirm what we have observed for α in Section 6.4. The acceptance ratio gradually decreases till a point and then it increases.

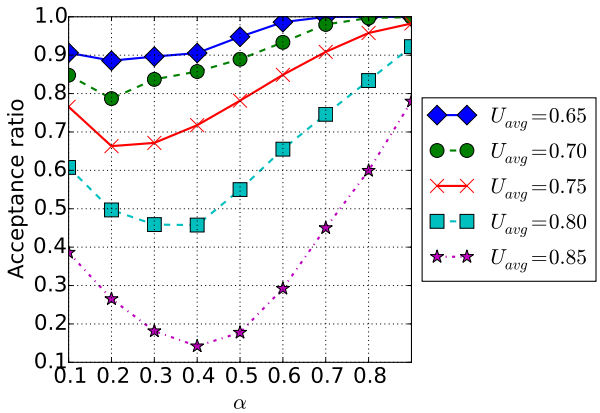

Figure 6.11: Impact of α

Chapter 7

Summary and Future Work

> Research is to see what everybody else has seen, and to think what nobody else has thought.
>
> Albert Szent-Gyorgyi

7.1 Summary and Conclusions

As we quoted in the epigraph of Chapter 1, "Almost all computer systems of the future will utilize real-time scientific principles and technology.", real-time systems and the systems that desire to apply real-time discipline are becoming ubiquitous with the advent of Internet of Things (IoTs) and Cyber-Physical Systems (CPS). The increasing complexity of real-time software and the emerging new hardware inspire us to revisit the “old-wise” in the embedded system community and the real-time community and to propose novel solutions dealing with the drastic changes in real-time systems.

The *HRT* scheduling framework proposed in [BS13] establishes a bridge between the data-flow models and the real-time theories, enabling us to directly apply real-time theories on the well-known data-flow models, e.g., SDF and CSDF. The *HRT* scheduling framework effectively converts actors in a CSDF graph into a periodic task set with implicit deadline and thus the majority of the theories developed in the real-time community can be applied to provide fast admission control and real-time guarantee for applications. However, the conversion done by the *HRT* scheduling framework comes at the cost of hurting the application latency which is one of the primary performance metrics for streaming applications. The proposers of *HRT* scheduling framework are aware of this issue [BS12] and suggest to select a smaller relative deadline for each

task to reduce the latency. In the real-time theories, such scaling down of deadlines of tasks negatively affects the schedulability of a multiprocessor real-time systems, and thus has to use more processors to compensate this negative effect, i.e., a larger number of processors are required to schedule the task set. The authors in [BS12] propose a simple way to uniformly select deadlines for all tasks but this approach is really ineffective in terms of resource minimization while meeting latency requirements. Therefore, to deal with this problem, in Chapter 3, we have proposed a new method to optimize the resource usage in the context of the *HRT* scheduling of CSDFs, where we formalize the resource minimization problem into an integer convex programming problem. By means of a off-the-shelf convex programming solver, we can obtain an optimal deadline selection for each task while minimizing the resource requirement and meanwhile ensuring the latency guarantee for CSDF-modeled streaming applications. The experimental results demonstrate the effectiveness of our approach over the existing approach in [BS12].

Due to the growing power consumption of increasingly complex applications, energy/power consumption is deemed as one of the major concerns when designing embedded systems. The single-ISA heterogeneous multicore systems are proposed to alleviate this energy pain. Nowadays such systems are prevalent in commercial electronic devices, such as mobile phones, TV boxs, etc. These systems provide designers a new opportunity to achieve energy efficiency and high performance and on the other hand they also require to find a new methodology to efficiently and effectively utilize the underlying hardware in an energy-efficient manner. Therefore, in Chapter 4, we have proposed a polynomial time algorithm to energy-efficiently map real-time streaming applications with latency and throughput constraints to cluster single-ISA heterogeneous multicore systems. Compared with existing approaches, the experimental results show that our proposed algorithm outperforms the existing approaches by finding a more energy efficient mapping. Our algorithm can save up to 34% energy on the cluster heterogeneous single-ISA multicore systems.

In Chapter 5, we have continued to study the problem of energy-efficient mapping on heterogeneous multicore systems. In this work, we have investigated the application of the C=D task-splitting [BDWZ12] on heterogeneous systems. We have analyzed and extended the C=D task-splitting for heterogeneous multicore systems. With our analysis and extension, we have proposed the ASHM algorithm to allocate and split real-time tasks on a heterogeneous multicore system. In contrast to fully partitioned allocation approaches, our proposed ASHM algorithm can effectively utilize energy-efficient cores to achieve more energy saving. The experimental results show the effectiveness of our proposed ASHM in terms of energy saving, where the maximum energy saving by ASHM compared to related approaches is up to 60%.

The trend towards integrating applications with different criticality levels on a

single HW/SW platform is emerging in safety-critical real-time systems. In order to satisfy the rigorous requirements of certification authorities and at the same time to better utilize the underlying HW/SW platform, a classical Mixed-Criticality (MC) model is proposed in [Ves07]. Although this classical MC model is able to capture the core features of MC systems, it receives criticism from system designers due to its pessimistic behavior of completely discarding all low critical application tasks when any high critical application task overruns. Imprecise MC (IMC) model is proposed in [BB13] to resolve the criticism, but its schedulability analysis under EDF-VD still was not studied. Therefore, in Chapter 6, we have studied the schedulability of the IMC model under EDF-VD and proposed a sufficient test. Based on the proposed sufficient test, we have derived a speedup factor function with respect to the utilization variation ratio α of all *high-criticality* tasks and the utilization variation ratio λ of all low-criticality tasks. This speedup factor function provides a good insight to observe the impact of α and λ on the speedup factor and enables us to quantify the optimality of EDF-VD for the IMC model in terms of speedup factor. Our experimental results show that our proposed sufficient test outperforms the existing AMC approach in terms of acceptance ratio. Moreover, the extensive experiments also confirm the observations we obtained for the speedup factor.

7.2 Future work

Although this dissertation has made several contributions to the real-time embedded system field, there remains interesting topics which can be researched based on our contributions. This section discusses some issues or challenges which deserve further investigation in the future.

7.2.1 The real convergence of data-flow models and real-time theories

An increasingly hot topic in the real-time community is the scheduling problem of parallel real-time directed acyclic graphs (DAG). We can clearly see the conceptual similarity between the DAG model used in the real-time community and the data-flow models used in the embedded system community. Considering the analogy of these models, it is worth to investigate how the DAG theories can be directly applied to data-flow models. Such directed application might be able to provide a better performance and real-time analysis framework and the conversion overhead occurred in the *HRT* scheduling framework, e.g., the increased latency (see in the problem studied in Chapter 3), might also be eliminated.

7.2.2 The multi-objective mapping of heterogeneous multicore systems

Our algorithms presented in Chapter 4 and 5 demonstrate the effectiveness in terms of energy efficiency. However, there are more objectives worth to be investigated in the complex HW/SW heterogeneous multicore, i.e., the thermal objective, the reliability objective and the security objective. These objectives interact with each other, thereby leaving us a large design space to exploit and requiring us to find a good trade-off between multiple objectives. Therefore, it is a very interesting and challenging problem to design an efficient and effective algorithm to map real-time applications onto a heterogeneous system with multiple objectives considered.

7.2.3 Practical and flexible MC model

Even though the IMC model has dealt successfully with some of the criticism from system designers, one assumption in the IMC model is still somehow pessimistic and in some cases impractical, i.e., if any *high*-criticality task overruns, all the other *high*-criticality tasks are assumed to overrun their smaller WCETs and thus be scheduled with their large WCETs (pessimistic ones). This assumption makes the system overreact to the overrun of a single *high*-criticality task and consequently it leads to an unnecessary degradation (i.e., reduced execution time) of *low*-criticality tasks. Therefore, further research can be conducted in the context of defining an MC model which can effectively reduce the pessimism of the current model and provide a more flexible execution semantics. To meet this goal, some existing theories, like control theories, might help.

Bibliography

[ARM16] ARMv8-A cores. https://www.arm.com/products/processors/cortex-a, Retrieved June 2016.

[ART14] ARTEMIS. http://www.artemis\-ju.eu/home_page, 2014.

[AS00] J. H. Anderson and A. Srinivasan. Pfair scheduling: beyond periodic task systems. In *Real-Time Computing Systems and Applications, 2000. Proceedings. Seventh International Conference on*, pages 297–306, 2000.

[AY03] Hakan Aydin and Qi Yang. Energy-aware partitioning for multiprocessor real-time systems. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, IPDPS '03, pages 113.2–, Washington, DC, USA, 2003. IEEE Computer Society.

[BB07] Theodore P. Baker and Sanjoy K. Baruah. Schedulability analysis of multiprocessor sporadic task systems. In *Handbook of Realtime and Embedded Systems*. CRC Press, 2007.

[BB13] Alan Burns and Sanjoy Baruah. Towards a more practical model for mixed criticality systems. In *Workshop on Mixed-Criticality Systems (colocated with RTSS)*, 2013.

[BBA10] Andrea Bastoni, Bjorn B. Brandenburg, and James H. Anderson. An empirical comparison of global, partitioned, and clustered multiprocessor edf schedulers. In *Proceedings of the 2010 31st IEEE Real-Time Systems Symposium*, RTSS '10, pages 14–24, Washington, DC, USA, 2010. IEEE Computer Society.

[BBA11] A. Bastoni, B. B. Brandenburg, and J. H. Anderson. Is semi-partitioned scheduling practical? In *2011 23rd Euromicro Conference on Real-Time Systems*, pages 125–135, July 2011.

[BBB+09] J. Barhorst, T. Belote, P. Binns, J. Hoffman, J. Paunicka, P. Sarathy, J. S. P. Stanfill, D. Stuart, and R. Urzi. White paper: A research agenda for mixed-criticality systems. http://www.cse.wustl.edu/~cdgill/CPSWEEK09_MCAR/, April 2009.

[BBD11] S. K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Proceedings of the 2011 IEEE 32Nd Real-Time Systems*

Symposium, RTSS '11, pages 34–43, Washington, DC, USA, 2011. IEEE Computer Society.

[BBD+12] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *Proceedings of the 2012 24th Euromicro Conference on Real-Time Systems*, ECRTS '12, pages 145–154, Washington, DC, USA, 2012. IEEE Computer Society.

[BBG16] S. Baruah, A. Burns, and Z. Guo. Scheduling mixed-criticality systems to guarantee some service under all non-erroneous behaviors. In *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 131–138, July 2016.

[BCPV93a] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 345–354, New York, NY, USA, 1993. ACM.

[BCPV93b] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 345–354, New York, NY, USA, 1993. ACM.

[BD15] Alan Burns and Robert Davis. Mixed criticality systems-a review. *University of York, Tech. Rep*, 2015.

[BDWZ12] A. Burns, R. I. Davis, P. Wang, and F. Zhang. Partitioned edf scheduling for multiprocessors using a c=d task splitting scheme. *Real-Time Systems*, 48(1):3–33, 2012.

[BELP96] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete. Cycle-static dataflow. *IEEE Transactions on Signal Processing*, 44(2):397–408, Feb 1996.

[BMAB16] Mario Bambagini, Mauro Marinoni, Hakan Aydin, and Giorgio Buttazzo. Energy-aware scheduling for real-time systems: A survey. *ACM Trans. Embed. Comput. Syst.*, 15(1):7:1–7:34, January 2016.

[BMR90] S. K. Baruah, A. K. Mok, and L. E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Real-Time Systems Symposium, 1990. Proceedings., 11th*, pages 182–190, Dec 1990.

[BRC06] P. Balbastre, I. Ripoll, and A. Crespo. Optimal deadline assignment for periodic real-time tasks in dynamic priority systems. In *18th Euromicro Conference on Real-Time Systems (ECRTS'06)*, pages 10 pp.–74, 2006.

[BS11] Mohamed Bamakhrama and Todor Stefanov. Hard-real-time scheduling of data-dependent tasks in embedded streaming applications. In *Proceedings of the Ninth ACM International Conference on Embedded Software*, EMSOFT '11, pages 195–204, New York, NY, USA, 2011. ACM.

[BS12] Mohamed A. Bamakhrama and Todor Stefanov. Managing latency in embedded streaming applications under hard-real-time scheduling. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '12, pages 83–92, New York, NY, USA, 2012. ACM.

[BS13] Mohamed A. Bamakhrama and Todor P. Stefanov. On the hard-real-time scheduling of embedded streaming applications. *Design Automation for Embedded Systems*, 17(2):221–249, 2013.

[But11] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer Publishing Company, Incorporated, 3rd edition, 2011.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[BZNS12] Mohamed A. Bamakhrama, Jiali Teddy Zhai, Hristo Nikolov, and Todor Stefanov. A methodology for automated design of hard-real-time embedded streaming systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '12, pages 941–946, San Jose, CA, USA, 2012. EDA Consortium.

[CGHJ09] J. Cong, K. Gururaj, G. Han, and W. Jiang. Synthesis algorithm for application-specific homogeneous processor networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(9):1318–1329, Sept 2009.

[CGJ97] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for np-hard problems. chapter Approximation Algorithms for Bin Packing: A Survey, pages 46–93. PWS Publishing Co., Boston, MA, USA, 1997.

[CHBK13] G. Chen, K. Huang, C. Buckl, and A. Knoll. Energy optimization with worst-case deadline guarantee for pipelined multiprocessor systems. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 45–50, March 2013.

[CHC+04] Jian-Jia Chen, Heng-Ruey Hsu, Kai-Hsiang Chuang, Chia-Lin Yang, Ai-Chun Pang, and Tei-Wei Kuo. Multiprocessor energy-efficient scheduling with task migration considerations. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, ECRTS '04, pages 101–108, Washington, DC, USA, 2004. IEEE Computer Society.

[Civ16] Civil Aviation Authority. `https://www.caa.co.uk/home/`, 2016.

[CK07] Jian-Jia Chen and Chin-Fu Kuo. Energy-efficient scheduling for real-time systems on dynamic voltage scaling (dvs) platforms. In *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, RTCSA '07, pages 28–38, Washington, DC, USA, 2007. IEEE Computer Society.

[CKR14] A. Colin, A. Kandhalu, and R. Rajkumar. Energy-efficient allocation of real-time applications onto heterogeneous processors. In *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 1–10, Aug 2014.

[CLL90] J.-Y. Chung, J.W.S. Liu, and Kwei-Jay Lin. Scheduling periodic jobs that allow imprecise results. *Computers, IEEE Transactions on*, 1990.

[CRJ06] Hyeonjoong Cho, Binoy Ravindran, and E. Douglas Jensen. An optimal real-time scheduling algorithm for multiprocessors. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, RTSS '06, pages 101–110, Washington, DC, USA, 2006. IEEE Computer Society.

[CST09] J. J. Chen, A. Schranzhofer, and L. Thiele. Energy minimization for periodic real-time tasks on heterogeneous processing units. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–12, May 2009.

[CT08] Jian-Jia Chen and Lothar Thiele. Energy-efficient task partition for periodic real-time tasks on platforms with dual processing elements. In *Proceedings of the 2008 14th IEEE International Conference on Parallel and Distributed Systems*, ICPADS '08, pages 161–168, Washington, DC, USA, 2008. IEEE Computer Society.

[CWLH08] T. Chantem, X. Wang, M. D. Lemmon, and X. S. Hu. Period and deadline selection for schedulability in real-time systems. In *2008 Euromicro Conference on Real-Time Systems*, pages 168–177, July 2008.

[CZZ+15] Hsiang-Yun Cheng, Jia Zhan, Jishen Zhao, Yuan Xie, Jack Sampson, and Mary Jane Irwin. Core vs. uncore: The heart of darkness. In *Proceedings of the 52Nd Annual Design Automation Conference*, DAC '15, pages 121:1–121:6, New York, NY, USA, 2015. ACM.

[DB11] Robert I. Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35:1–35:44, October 2011.

[Der74] ML Dertouzos. Control robotics: the procedural control of physical processes," information processing 74, 1974.

[DGR+74] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, Oct 1974.

[DZDN+07] Abhijit Davare, Qi Zhu, Marco Di Natale, Claudio Pinello, Sri Kanajan, and Alberto Sangiovanni-Vincentelli. Period optimization for hard real-time distributed automotive systems. In *Proceedings of the 44th Annual Design Automation Conference*, DAC '07, pages 278–283, New York, NY, USA, 2007. ACM.

[Eas13] Arvind Easwaran. Demand-based scheduling of mixed-criticality sporadic tasks on one processor. In *Real-Time Systems Symposium (RTSS), 2013 IEEE 34th*, pages 78–87. IEEE, 2013.

[EBSA+11] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 365–376, New York, NY, USA, 2011. ACM.

[EET04] EETimes. Intel cancels tejas, moves to dual-core designs. `http://www.eetimes.com/document.asp?doc_id=1150169`, May 2004.

[ENNT15] Alexandre Esper, Geoffrey Nelissen, Vincent Nélis, and Eduardo Tovar. How realistic is the mixed-criticality real-time system model? In *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, RTNS '15, pages 139–148, New York, NY, USA, 2015. ACM.

[ESAS14] Abdullah Elewi, Mohamed Shalan, Medhat Awadalla, and Elsayed M. Saad. Energy-efficient task allocation techniques for asymmetric multiprocessor embedded systems. *ACM Trans. Embed. Comput. Syst.*, 13(2s):71:1–71:27, January 2014.

[EY14] Pontus Ekberg and Wang Yi. Bounding and shaping the demand of generalized mixed-criticality sporadic task systems. *Real-time systems*, 50(1):48–86, 2014.

[Fed16] Federal Aviation Administration. `http://www.faa.gov/`, 2016.

[GB08] Michael Grant and Stephen Boyd. Graph implementations for nonsmooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008.

[GB14] Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. `http://cvxr.com/cvx`, March 2014.

[GBK+12] Vishal Gupta, Paul Brett, David Koufaty, Dheeraj Reddy, Scott Hahn, Karsten Schwan, and Ganapati Srinivasa. The forgotten 'uncore': On the energy-efficiency of heterogeneous cores. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, pages 34–34, Berkeley, CA, USA, 2012. USENIX Association.

[Gil15] Lori Gil. Nvidia's tegra x1 crushes the competition. `http://liliputing.com/2015/02/nvidias-tegra-x1-crushes-the-competition.html`, Mar 2015.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[GSYY10] Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. Fixed-priority multiprocessor scheduling with liu and layland's utilization bound. In *Proceedings of the 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS '10, pages 165–174, Washington, DC, USA, 2010. IEEE Computer Society.

[HCH11] S. Hong, T. Chantem, and X. S. Hu. Meeting end-to-end deadlines through distributed local deadline assignments. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 183–192, Nov 2011.

[HGST14] P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele. Service adaptions for mixed-criticality systems. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 125–130, Jan 2014.

[HKPS15] Jörg Henkel, Heba Khdr, Santiago Pagani, and Muhammad Shafique. New trends in dark silicon. In *Proceedings of the 52Nd Annual Design Automation Conference*, DAC '15, pages 119:1–119:6, New York, NY, USA, 2015. ACM.

[HM07] Sebastian Herbert and Diana Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Proceedings of the 2007 International Symposium on Low Power Electronics and Design*, ISLPED '07, pages 38–43, New York, NY, USA, 2007. ACM.

[HTC07] T. Y. Huang, Y. C. Tsai, and E. T. H. Chu. A near-optimal solution for the heterogeneous multi-processor single-level voltage setup problem. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pages 1–10, March 2007.

[J+13] Mathieu Jan et al. Maximizing the execution rate of low-criticality tasks in mixed criticality system. In *Proceddings of Workshop of Mixed-Criticality (WMC), Real-Time Systems Symposium (RTSS)*, 2013.

[Jef12] Brain Jeff. Advances in big.little technology for power and energy savings. Technical report, ARM Ltd, Sept 2012.

[JHIP10] H. Javaid, X. He, A. Ignjatovic, and S. Parameswaran. Optimal synthesis of latency and throughput constrained pipelined mpsocs targeting streaming applications. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2010 IEEE/ACM/IFIP International Conference on*, pages 75–84, Oct 2010.

[JLBK13] J. Augusto Santos Júnior, George Lima, Konstantinos Bletsas, and Shinpei Kato. Multiprocessor real-time scheduling with a few migrating tasks. In *Proceedings of the 2013 IEEE 34th Real-Time Systems Symposium*, RTSS '13, pages 170–181, Washington, DC, USA, 2013. IEEE Computer Society.

[Joh74] David S. Johnson. Fast algorithms for bin packing. *J. Comput. Syst. Sci.*, 8(3):272–314, jun 1974.

[JSM91] K. Jeffay, D. F. Stanat, and C. U. Martel. On non-preemptive scheduling of period and sporadic tasks. In *Real-Time Systems Symposium, 1991. Proceedings., Twelfth*, pages 129–139, Dec 1991.

[KAB+03] Nam Sung Kim, Todd Austin, David Blaauw, Trevor Mudge, Krisztián Flautner, Jie S. Hu, Mary Jane Irwin, Mahmut Kandemir, and Vijaykrishnan Narayanan. Leakage current: Moore's law meets static power. *Computer*, 36(12):68–75, December 2003.

[KFJ+03] Rakesh Kumar, Keith I. Farkas, Norman P. Jouppi, Parthasarathy Ranganathan, and Dean M. Tullsen. Single-isa heterogeneous multi-core architectures: The potential for processor power reduction. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 81–, Washington, DC, USA, 2003. IEEE Computer Society.

[KT51] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, Berkeley, Calif., 1951. University of California Press.

[KYD11] F. Kong, W. Yi, and Q. Deng. Energy-efficient scheduling of real-time tasks on cluster-based multicores. In *2011 Design, Automation Test in Europe*, pages 1–6, March 2011.

[Leu89] Joseph Y-T Leung. A new algorithm for scheduling periodic, real-time tasks. *Algorithmica*, 4(1-4):209–219, 1989.

[LG11] J. Lu and Y. Guo. Energy-aware fixed-priority multi-core scheduling for real-time systems. In *2011 IEEE 17th International Conference on Embedded and Real-Time Computing Systems and Applications*, volume 1, pages 277–281, Aug 2011.

[LL73] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973.

[LLS+91] Jane W.-S. Liu, Kwei-Jay Lin, Wei Kuan Shih, Albert Chuang-shi Yu, Jen-Yao Chung, and Wei Zhao. Algorithms for scheduling imprecise computations. In *Foundations of Real-Time Computing: Scheduling and Resource Management*, pages 203–249. Springer, 1991.

[LM87] Edward Ashford Lee and David G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. Comput.*, 36(1):24–35, January 1987.

[LPG+14] J. Lee, K. M. Phan, X. Gu, J. Lee, A. Easwaran, I. Shin, and I. Lee. Mc-fluid: Fluid model-based mixed-criticality scheduling on multiprocessors. In *Real-Time Systems Symposium (RTSS), 2014 IEEE*, pages 41–52, Dec 2014.

[LSCS15] D. Liu, J. Spasic, G. Chen, and T. Stefanov. Energy-efficient mapping of real-time streaming applications on cluster heterogeneous mpsocs. In *Embedded Systems For Real-time Multimedia (ESTIMedia), 2015 13th IEEE Symposium on*, pages 1–10, Oct 2015.

[LSG+16] D. Liu, J. Spasic, N. Guan, G. Chen, S. Liu, T. Stefanov, and W. Yi. Edf-vd scheduling of mixed-criticality systems with degraded quality guarantees. In *2016 IEEE Real-Time Systems Symposium (RTSS)*, pages 35–46, Nov 2016.

[LSL+94] J. W. S. Liu, Wei-Kuan Shih, Kwei-Jay Lin, R. Bettati, and Jen-Yao Chung. Imprecise computations. *Proceedings of the IEEE*, 82(1):83–94, Jan 1994.

[MB07] Orlando M. Moreira and Marco J. G. Bekooij. Self-timed scheduling analysis for real-time applications. *EURASIP Journal on Advances in Signal Processing*, 2007(1):083710, 2007.

[Mit15] Tulika Mitra. Heterogeneous multi-core architectures. *Information and Media Technologies*, 10(3):383–394, 2015.

[Mit16] Sparsh Mittal. A survey of techniques for architecting and managing asymmetric multicore processors. *ACM Computing Surveys*, 2016.

[Nor] Sven Nordhoff. Do-178c/ed-12c. `https://www.sqs.com/nl/_download/DO-178C_ED-12C.pdf`.

[ODR16] ODROID. `http://www.hardkernel.com/`, 2016.

[RKKK14a] RC Ravindran, C Mani Krishna, Israel Koren, and Zahava Koren. Scheduling imprecise task graphs for real-time applications. *International Journal of Embedded Systems*, 6(1):73–85, 2014.

[RKKK14b] R.C. Ravindran, C. Mani Krishna, Israel Koren, and Zahava Koren. Scheduling imprecise task graphs for real-time applications. *International Journal of Embedded Systems (IJES)*, 6, 2014.

[Sam16] Samsung Exynos. `http://www.samsung.com/`, 2016.

[SDK13] Amit Kumar Singh, Anup Das, and Akash Kumar. Energy optimization by exploiting execution slacks in streaming applications on multiprocessor systems. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 115:1–115:7, New York, NY, USA, 2013. ACM.

[SGB06] S. Stuijk, M.C.W. Geilen, and T. Basten. SDF^3: SDF For Free. In *Application of Concurrency to System Design, 6th International Conference, ACSD 2006, Proceedings*, pages 276–278. IEEE Computer Society Press, Los Alamitos, CA, USA, June 2006.

[SGTG12] F. Santy, L. George, P. Thierry, and J. Goossens. Relaxing mixed-criticality scheduling strictness for task sets scheduled with fp. In *Proceedings of the 24th Euromicro Conference on Real-Time Systems*, July 2012.

[SGZ14] Hang Su, Nan Guan, and Dakai Zhu. Service guarantee exploration for mixed-criticality systems. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014 IEEE 20th International Conference on*, pages 1–10, Aug 2014.

[SLCS16] Jelena Spasic, Di Liu, Emanuele Cannella, and Todor Stefanov. On the improved hard real-time scheduling of cyclo-static dataflow. *ACM Trans. Embed. Comput. Syst.*, 15(4):68:1–68:26, August 2016.

[SLS16] J. Spasic, D. Liu, and T. Stefanov. Exploiting resource-constrained parallelism in hard real-time streaming applications. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 954–959, March 2016.

[Sta88] John A. Stankovic. Misconceptions about real-time computing: A serious problem for next-generation systems. *Computer*, 21(10):10–19, October 1988.

[SZ13] Hang Su and Dakai Zhu. An elastic mixed-criticality task model and its scheduling algorithm. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, pages 147–152, San Jose, CA, USA, 2013. EDA Consortium.

[TA10] William Thies and Saman Amarasinghe. An empirical characterization of stream programs and its implications for language and compiler design. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, PACT '10, pages 365–376, New York, NY, USA, 2010. ACM.

[Tho12] Haydn Thompson. Mixed criticality systems. `http://cordis.europa.eu/fp7/ict/embedded-systems-engineering/presentations/thompson.pdf`, February 2012.

[Ves07] Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, RTSS '07, pages 239–243, Washington, DC, USA, 2007. IEEE Computer Society.

[WLL+11] Yi Wang, Hui Liu, Duo Liu, Zhiwei Qin, Zili Shao, and Edwin H.-M. Sha. Overhead-aware energy optimization for real-time streaming applications on multiprocessor system-on-chip. *ACM Trans. Des. Autom. Electron. Syst.*, 16(2):14:1–14:32, apr 2011.

[XMM07] Ruibin Xu, Rami Melhem, and Daniel Mosse. Energy-aware scheduling for streaming applications on chip multiprocessors. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, RTSS '07, pages 25–38, Washington, DC, USA, 2007. IEEE Computer Society.

[ZB09] Fengxiang Zhang and Alan Burns. Schedulability analysis for real-time systems with edf scheduling. *IEEE Trans. Comput.*, 58(9):1250–1258, September 2009.

[ZBS13] Jiali Teddy Zhai, Mohamed A. Bamakhrama, and Todor Stefanov. Exploiting just-enough parallelism when mapping streaming applications in hard real-time systems. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 170:1–170:8, New York, NY, USA, 2013. ACM.

Appendix

Appendix I

Lemma 1. The minimum value of piece-wise function (6.22) given in Section 6.4 is obtained when $b = b_0$.

$$s(b) = \begin{cases} \frac{(\alpha\lambda^2-\alpha\lambda)b^2+b-1}{(\alpha\lambda-\alpha+1)b-1} & 0 < b \leq b_0 \\ \frac{(1-\alpha)b^2+(\alpha\lambda+\alpha-1)b-\alpha}{(\alpha\lambda-\alpha+1)b-1} & b_0 < b \leq 1 \end{cases} \tag{1}$$

Proof. For case of $0 < b \leq b_0$, its derivative is

$$s'(b) = \frac{\alpha(\lambda-1)(\lambda(\alpha y-\alpha+1)b^2-2\lambda b+1)}{((\alpha\lambda-\alpha+1)b-1)^2}$$

The denominator is obviously positive. For the numerator, since the discriminant of $\lambda(\alpha\lambda-\alpha+1)b^2-2\lambda b+1=0$ is $(2\lambda)^2-4\lambda(\alpha\lambda-\lambda+1)$, which is negative since $0<\lambda<1$, so we know $\lambda(\alpha\lambda-\alpha+1)b^2-2\lambda b+1>0$. Moreover, we have $\lambda-1<0$, so putting them together we know the numerator is negative. In summary, $s'(b)$ is negative and thus $s(b)$ is monotonically decreasing with respect to b in the range $b \in (0, b_0]$.

For case of $b_0 < b \leq 1$, we can compute the derivative of $s(b)$ by

$$s'(b) = \frac{(1-\lambda)((\lambda y-x+1)b^2-2b-(\lambda y-x-1))}{((\lambda y-x+1)b-1)^2}$$

The denominator is obviously positive. For the numerator, we focus on $(x\lambda-x+1)b^2-2b-(x\lambda-x-1)$ part. The following equation

$$(x\lambda-x+1)b^2-2b-(x\lambda-x-1)=0$$

has two roots $b_1=1$ and $b_2=\frac{1+(x-x\lambda)}{1-(x-x\lambda)}$, which is greater than 1, so we know $(x\lambda-x+1)b^2-2b-(x\lambda-x-1)$ is either always positive or always negative in the range

of $b \in (b_0, 1)$. Since we can construct $(x\lambda - x + 1)b^2 - 2b - (x\lambda - x - 1) > 0$ with $x = \lambda = b = 0.5$, so we know $(x\lambda - x + 1)b^2 - 2b - (x\lambda - x - 1)$ is always positive. Moreover, since $1 - x > 0$, the numerator of $s'(b)$ is positive, so overall $s'(b)$ is positive, and thus $s(b)$ is monotonically increasing with respect to b in the range of $b \in (b_0, 1]$.

In summary, we have proved $s(b)$ is monotonically decreasing in $(0, b_0]$, and monotonically increasing in $(b_0, 1]$, both with respect to b, so the smallest value of $s(b)$ must occur at b_0. □

Lemma 2. If $0 < \alpha < 1$ and $0 \leq \lambda < 1$, then

$$b_0^1 = \frac{(2 - \alpha\lambda - \alpha) + (1 - \lambda)\sqrt{-3\alpha^2 + 4\alpha}}{2(-\alpha\lambda^2 + \alpha\lambda - \alpha + 1)} > 1 \tag{2}$$

$$b_0^2 = \frac{(2 - \alpha\lambda - \alpha) - (1 - \lambda)\sqrt{-3\alpha^2 + 4\alpha}}{2(-\alpha\lambda^2 + \alpha\lambda - \alpha + 1)} \in [0, 1] \tag{3}$$

Proof. We start with proving $b_0^1 > 1$. We first prove $b_0^1 \geq 0$ by showing both the numerator and dominator are positive. For simplicity, we use N_1 and M_1 to denote the numerator and denominator of b_0^1 in (2), and N_2 and M_2 the numerator and denominator of b_0^2 in (3). Note that the following reasoning relies on that $\alpha \in (0, 1), \lambda \in [0, 1)$.

1. $N_1 > 0$. First, we have

$$\begin{aligned} & N_1 \times N_2 \\ &= (2 - \alpha\lambda - \alpha)^2 - (1 - \lambda)^2(-3\alpha^2 + 4\alpha) \\ &= 4\alpha\lambda(1 - \lambda)(1 - \alpha) + 4(1 - \alpha)^2 \\ &> 0 \end{aligned}$$

   Moreover, it is easy to see $N_2 > 0$. Therefore, we can conclude that N_1 is also positive.

2. $M_1 > 0$. $2(-\alpha\lambda^2 + \alpha\lambda - \alpha + 1) = 2(\alpha\lambda(1 - \lambda) + (1 - \alpha))$, which is positive.

In summary, both the numerator and the denominator of b_0^1 in (2) are positive, so $b_0^1 \geq 0$. Next we prove $b_0^1 \leq 1$ by showing $N_1 - M_1 \leq 0$:

$$\begin{aligned} & N_1 - M_1 \\ &= (\lambda - 1)(\sqrt{-3\alpha^2 + 4\alpha} + \alpha(2\lambda - 1)) \end{aligned}$$

which is negative if $\lambda \geq 0.5$ (since $\lambda - 1 < 0$ and $\sqrt{-3\alpha^2 + 4\alpha} + \alpha(2\lambda - 1) \geq 0$). So in the following we focus on the case of $\lambda < 0.5$. Since $\lambda < 0.5$, we know $\alpha(2\lambda - 1)$

is negative, so we define two positive number A and B as follows

$$A = \sqrt{-3\alpha^2 + 4\alpha} \tag{4}$$
$$B = \alpha(1 - 2\lambda) \tag{5}$$

so $N_1 - M_1 = (\lambda - 1)(A - B)$. Since $\lambda - 1 < 0$, we only need to prove $A - B > 0$, which is equivalent to proving $A^2 - B^2 > 0$ (as both A and B are positive): $A^2 - B^2 > 0$, which is done as follows:

$$\begin{aligned} A^2 - B^2 =& -3\alpha^2 + 4\alpha - \alpha^2(2\lambda - 1)^2 \\ =& 4\alpha(1 - \alpha) + 4\alpha^2\lambda(1 - \lambda) \\ >& 0 \end{aligned}$$

so we have $A - B > 0$ and thus $N_1 - M_1 = (\lambda - 1)(A - B) < 0$. In summary, we have proved $N_1 - M_1 < 0$ for the cases of both $\lambda \geq 0.5$ and $\lambda < 0.5$, so we know $b_0^1 \in [0, 1]$.

Next we prove $b_0^2 > 1$, by showing $N_2 - M_2 > 0$

$$\begin{aligned} & N_2 - M_2 \\ =& (1 - \lambda)(\sqrt{-3\alpha^2 + 4\alpha} - \alpha(2\lambda - 1)) \end{aligned}$$

If $\lambda \leq 0.5$, then $\sqrt{-3\alpha^2 + 4\alpha} - \alpha(2\lambda - 1) > 0$, and since $1 - \lambda > 0$ we have $N_2 - M_2 > 0$. If $\lambda > 0.5$, we let $C = \alpha(2\lambda - 1) > 0$ and also use A as defined above, $N_2 - M_2 = (1 - \lambda)(A - C)$. To prove $A - C > 0$, it suffices to prove $A^2 - C^2 > 0$, as shown in the following:

$$\begin{aligned} A^2 - C^2 =& -3\alpha^2 + 4\alpha - \alpha^2(2\lambda - 1)^2 \\ =& 4\alpha - (3 + (2\lambda - 1)^2)\alpha^2 \\ >& 4\alpha - 4\alpha^2 \ \ (\lambda < 1 \text{ ,so } 2\lambda - 1 < 1) \\ >& 0 \end{aligned}$$

By now we have proved $N_2 - M_2$ for both cases of $\lambda \leq 0.5$ and $\lambda > 0.5$, so we known $b_0^2 > 1$.

□

Appendix II

Experimental results between EDF-VD and AMC are depicted in Figure 1 - 3, where pCriticality$= 0.3$.

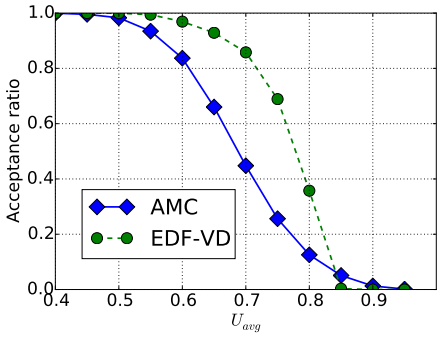

Figure 1: $\lambda = 0.3$

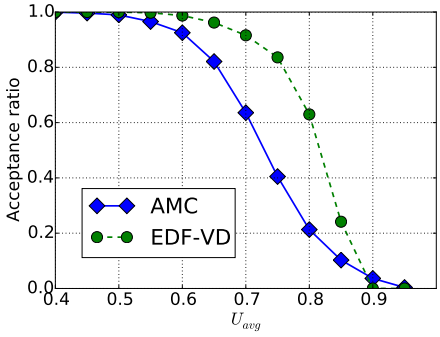

Figure 2: $\lambda = 0.5$

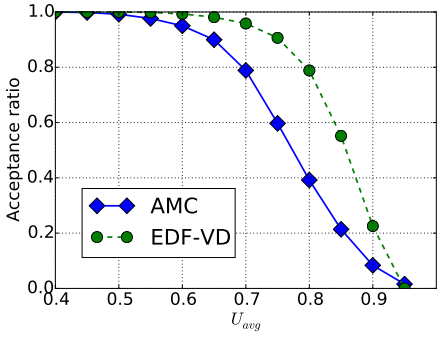

Figure 3: $\lambda = 0.7$

List of Publications

First Author Publications

1. **Di Liu**, Jelena Spasic, Jiali Teddy Zhai, Gang Chen, and Todor Stefanov, "Resource Optimization for CSDF-modeled Streaming Applications with Latency Constraints", In Proc. "17th Int. Conf. Design, Automation and Test in Europe (DATE'14)", pp. 1-6 , Dresden, Germany, Mar. 24-28, 2014

2. **Di Liu**, Jelena Spasic, Gang Chen, and Todor Stefanov, "Energy-Efficient Mapping of Real-Time Streaming Applications on Cluster Heterogeneous MPSoCs", In Proc. "13th Int. IEEE Symposium on Embedded Systems for Real-Time Multimedia (ESTIMedia'15)", pp. 1-10, Amsterdam, The Netherlands, Oct. 8-9, 2015.

3. **Di Liu**, Jelena Spasic, Peng Wang, and Todor Stefanov, "Energy-Efficient Scheduling of Real-Time Tasks on Heterogeneous Multicores Using Task Splitting", In Proc. "22nd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'16)", pp. 1-10 , Daegu, South Korea, Aug. 17-19, 2016.

4. **Di Liu**, Jelena Spasic, Nan Guan, Gang Chen , Songran Liu, Todor Stefanov, Wang Yi, "EDF-VD Scheduling of Mixed-Criticality Systems with Degraded Quality Guarantees", in Proc. "2016 The IEEE Real-Time Systems Symposium (RTSS'16)", pp. 35-46 , Porto, Portugal, Nov. 29 - Dec. 02, 2016

Co-author Publications

1. Gang Chen, Biao Hu, Kai Huang, Alois Knoll, **Di Liu**, "Abstract: Shared L2 Cache Management in Multicore Real-Time System," 2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 170-170, Boston, USA, 11-13 May 2014

2. Gang Chen, Biao Hu, Kai Huang, Alois Knoll, **Di Liu**, and Todor Stefanov, "Automatic Cache Partitioning and Time-triggered Scheduling for Real-time MPSoCs", In Proc. "2014 International Conference on Reconfigurable Computing and FPGAs (ReConFig 2014)", pp. 1-8., Cancun, Mexico, Dec. 8-10, 2014.

3. Jelena Spasic, **Di Liu**, Emanuele Cannella, and Todor Stefanov, "Improved Hard Real-Time Scheduling of CSDF-modeled Streaming Applications", In Proc. "IEEE/ACM/IFIP Int. Conf. on HW/SW Codesign and System Synthesis (CODES+ISSS'15)", pp. 65-74, Amsterdam, The Netherlands, Oct. 4-9, 2015.

4. Jelena Spasic, **Di Liu**, Emanuele Cannella, and Todor Stefanov, "Exploiting Resource-constrained Parallelism in Hard Real-Time Streaming Applications", In Proc. "19th Int. Conf. Design, Automation and Test in Europe (DATE'16)", pp. 954-959, Dresden, Germany, Mar. 14-18, 2016.

5. Jelena Spasic, **Di Liu**, Emanuele Cannella, and Todor Stefanov, "On the Improved Hard Real-Time Scheduling of Cyclo-Static Dataflow", ACM Transactions on Embedded Computing Systems (TECS), vol. 15, Issue 4, Article 68, Aug 2016

6. Jelena Spasic, **Di Liu**, and Todor Stefanov, "Energy-Efficient Mapping of Real-Time Applications on Heterogeneous MPSoCs using Task Replication", In Proc. "IEEE/ACM/IFIP Int. Conf. on HW/SW Codesign and System Synthesis (CODES+ISSS'16)", pp. 1-10, Pittsburgh, USA, Oct. 2-7, 2016.

Index

Samenvatting

Systemen worden real-time systemen genoemd wanneer de correctheid van het systeem niet enkel afhangt van de juistheid van de systeemoutput, maar ook van het feit of de output tijdig geleverd kan worden. Medische systemen, voertuigen, luchtvaartuigen enz. zijn voorbeelden van real-time systemen. Door de komst van 'Het Internet der Dingen' (IoT) en Cyber-fysieke Systemen (CPS) zijn real-time systemen en systemen die een real-time inbreng vereisen alomtegenwoordig. De stijgende complexiteit van real-time software en de opkomst van nieuwe hardware zetten ons aan om even stil te staan bij de bekende embeded system community en de real-time community, en om nieuwe oplossingen betreffende drastische veranderingen op het vlak van real-time systemen voor te stellen. Daarom stellen we in dit proefschrift nieuwe technieken en algoritmen voor om de prestaties betreffende vertraging, energie en schedulability van real-time systemen te verbeteren.

In het eerste deel ligt de focus vooral op de optimalisatie van de vertraging van real-time applicaties die uitgedrukt worden in cyclo-statische-dataflow (CSDF) grafieken. Binnen de context van een hard-real-time planningskader wordt de vertraging van een door een CSDF gemodelleerde real-time applicatie beïnvloed door het selecteren van een passende deadline voor elk knooppunt in de CSDF-grafiek. Hierbij wordt de deadline gebruikt om te controleren of de berekening van de werklast tijdig wordt uitgevoerd. Naast vertraging heeft de deadline-sectie ook een invloed op de noodzakelijke middelen voor de planning van de CSDF gemodelleerde real-time applicatie. Daarom moeten de deadlines van CSDF-knooppunten zorgvuldig geselecteerd worden, zodat aan de beoogde vertraging van de applicatie voldaan kan worden en gelijktijdig de middelen, die de applicatie vereist, verminderd kunnen worden. Wij bewijzen dat, met het paradigma van de globale planning, het probleem van de deadlineselectie geformuleerd kan worden als een integraal convex programmeerprobleem. Aan de hand van de pasklare convexe programmeeroplosser kan dit probleem optimaal opgelost worden.

In het tweede deel ligt de focus op de energie-optimalisatie van real-time applicaties of heterogene multicore-systemen. Heterogene multicore-systemen nemen gaandeweg de plaats in van de traditionele homogene multicore-systemen om zo de

energie-efficiëntie van systemen te verhogen. De energieconsumptie van real-time applicaties kan verminderd worden door de berekening van de werklast van real-time applicaties toe te kennen aan passende kernen terwijl de prestatie gegarandeerd blijft.

In het laatste deel ligt de focus op het garanderen van real-time beperkingen van 'mixed-criticality'-systemen – een specifiek type van real-time systemen – waarbij real-time applicaties met een verschillende belangrijkheidsgraad uitgevoerd worden op een gedeeld hardware-platform. Zo is bijvoorbeeld het infotainment in een voertuig minder belangrijk dan de besturingsapplicatie van de auto en kunnen beide via hetzelfde verwerkingsplatform werken om zo de grootte, het gewicht en de kracht van het systeem te verminderen. Wij stellen een beknopte test voor om de real-time beperkingen van zulke 'mixed-criticality'-systemen effectief en efficiënt te testen.

Acknowledgements

It would not have been possible for me to approach the end of the difficult Ph.D journey without numerous support and help I received during the past six years.

First of all, I'd like to thank China Scholarship Council for sponsoring me to pursue my Ph.D in Leiden University.

My unique and big 'hvala' (thank) goes to **Jelena Spasic**. She exemplifies what an excellent roommate would look like, and I was really fortunate to have her as my roommate since the very first day of my Ph.D until the end. She is really helpful and supportive whenever I need a favor. Academically, she is a diligent and smart girl. I benefit a lot from our technical brainstorm, through which many of my raw ideas are concretized, refined, and eventually distilled into the final publications. Additionally, I would like to show my gratitude to **Milos Acanski**, Jelena's husband, for the great times we had together. I believe that the friendship between us will never be evanescent regardless of geo-distance.

Working in the 'United Nation' (the embedded system group) was a pleasure experience because of my colleagues. **Mohammad Al Hissi**, **Mohamed Bamakhrama**, **Emanuele Cannella**, and **Teddy Zhai**, I really appreciate the assistance and advice I obtained from you guys, which paved the way for me to establish my inceptively academic insight. Also it has been a great pleasure to work with **Tsvetan Shoshkov**, **Hristo Nikolov**, and **Sven van Haastregt**, thank you. **Sobhan Niknam, Peng Wang, Hongchang Shan, Christian Fuchs**, and **Erqian Tang**, our new colleagues in the group, I wish you to go well with your own Ph.D journey.

I want to thank the following friends in LIACS for so many interesting talks and great moments. **Song Wu, Yu Liu, Yanming Guo, Fuyu Cai, Yuanhao Guo, Xiaoqing Tang, Zhan Xiong, Hao Wang, Kaifeng Yang, Zhiwei Yang, Bilal Karasneh, Mohamed Tleis** and **Mohd Hafeez Osman**.

In Leiden, I have been really lucky to make many terrific friends. I am grateful to **Shenfa Miao** for his help in many ways. I cannot remember how many times I knocked his door asking for help, never rejected. My appreciation also goes to Shenfa's family and thank them for being the friends of my wife and daughter. **Peng Ye, Chenjie Xing, Jiaqi Zhao, Yang Liu, Zhiguo Zhou, Hui Liu, Jianbing Jiang,**

Wei Wang, Yaowang Li, Bo Wang, and **Yihao Li**. I would like to thank you for the help and happy times we had together.

I sincerely appreciate **Dr. Nan Guan** from Hong Kong Polytechnic University for offering a five-month visit to his group. Although the visit is transitory, I am impressed by his broad knowledge and benefited a lot from his kind guidance. This experience will definitely have a long-term impact on my career. My HK life was fulfilled with joy and laughter because of some new friends, **Xu Jiang, Zhu Wang, Jinghao Sun, Yue Tang, Tao Yang, Xuxuan Zhou, Jun Wang**, and **Yujun Fu**, thank you. I believe our friendship will never fade away.

I want to express the gratitude to my grow-up friend, **Zhuozhou Qin**. So many times, he came to the airport and picked me up no matter how later my flight was. A couple also gets my special thank, **Qi Huang** and **Qian Ding** (my dear panda). They always encouraged me to insist during the difficult Ph.D journey and provided many precious advice. In addition, I would like to thank them for their 'luxury' schedule, when I visited San Diego. When chatting in the living room, I really felt like that we were back to NWPU ten years ago.

Finally, I want to thank my family member for their unselfish support and patience, my brother-in-law **Deqiang Huang**, my nephew **Chen Huang**, my mother-in-law **Yonghong Zhang**, and my father-in-law **Chugang Xiong**. I am sincerely indebted to my sister **Yun Liu**, my mother **Shixiang Yang**, and my father **Wenguang Liu**. Without their unconditional support and continuous love, this thesis would not have been possible. In particular, I am very very grateful to my wife **Yan Liu** for being patient and kind to such an anxious PhD student. It was not an easy decision to start an unpredictable oversea journey with me, thank you for the sacrifice, love, and support. My little angel, **Ruolai Liu**, thank you for bringing many joyful times to our family. Your smiling face is my source of the courage, enabling me to overcome any difficulties during the journey.

Curriculum Vitae

Di Liu was born on October 25, 1984 in Lijiang, Yunnan, China. He obtained his B.Eng degree and M.Eng degree from Northwestern Polytechnical University, China in 2011. After his graduation, he joined the Leiden Embedded Research Center at Leiden University as a PhD student, where his research work was mainly sponsored by China Scholarship Council. From November 2016 till April 2017, he worked as a research assistant in the Real-Time System group of Hong Kong Polytechnic University, Hong Kong.