


6-26-2015

On the Design of Real-Time Systems on Multi-Core Platforms under Uncertainty

TIANYI WANG

Florida International University, tiawang@fiu.edu

Follow this and additional works at: <http://digitalcommons.fiu.edu/etd>

 Part of the [Computer and Systems Architecture Commons](#), and the [Electrical and Computer Engineering Commons](#)

Recommended Citation

WANG, TIANYI, "On the Design of Real-Time Systems on Multi-Core Platforms under Uncertainty" (2015). *FIU Electronic Theses and Dissertations*. Paper 2219.

<http://digitalcommons.fiu.edu/etd/2219>

This work is brought to you for free and open access by the University Graduate School at FIU Digital Commons. It has been accepted for inclusion in FIU Electronic Theses and Dissertations by an authorized administrator of FIU Digital Commons. For more information, please contact dcc@fiu.edu.

FLORIDA INTERNATIONAL UNIVERSITY
Miami, Florida

ON THE DESIGN OF REAL-TIME SYSTEMS ON MULTI-CORE
PLATFORMS UNDER UNCERTAINTY

A dissertation submitted in partial fulfillment of the
requirements for the degree of
DOCTOR OF PHILOSOPHY
in
ELECTRICAL ENGINEERING
by
Tianyi Wang

2015

To: Dean Amir Mirmiran
College of Engineering and Computing

This dissertation, written by Tianyi Wang, and entitled On the Design of Real-Time Systems on Multi-core Platforms under Uncertainty, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

Kang K. Yen

Jean H. Andrian

Nezih Pala

Deng Pan

Gang Quan, Major Professor

Date of Defense: June 26, 2015

The dissertation of Tianyi Wang is approved.

Dean Amir Mirmiran
College of Engineering and Computing

Dean Lakshmi N. Reddi
University Graduate School

Florida International University, 2015

© Copyright 2015 by Tianyi Wang

All rights reserved.

DEDICATION

To my mom.

ACKNOWLEDGMENTS

First of all, I would like to thank my major advisor, Dr. Gang Quan for his guidance, time, and patience during my entire doctoral study. I truly admire his scientific dedication and rigorous attitude to searching for the truth which will inspire me for the rest of my life.

Next, I would also like to express my gratitude to my Ph.D. committee members, Dr. Jean H. Andrian, Dr. Kang Yen, Dr. Nezhil Pala, and Dr. Deng Pan, for their helpful insights, comments, and suggestions in improving the quality of this dissertation. It is truly honored to have such great fantastic and knowledgeable professors serving as my committee members.

Further, I would like to thank my lab mates, Mr. Gustavo Chaparro, Mr. Qiushi Han, Mr. Soamar Homsil, Mr. Shi Sha, Mr. Shuo Liu, Dr. Ming Fan, Dr. Vivek Chaturvedi, Dr. Guanglei Liu, and Dr. Huang Huang for creating an amazing working environment.

Additionally, I would like to thank the National Science Foundation (NSF). This dissertation is supported in part by NSF number projects CNS-0746643, CNS-0917021, CNS-0969013, CNS-1018108, CNS-1018731, and CNS-1423137.

Finally, I want to thank my family and my friends for their unconditional love, faith, and encouragement.

ABSTRACT OF THE DISSERTATION
ON THE DESIGN OF REAL-TIME SYSTEMS ON MULTI-CORE
PLATFORMS UNDER UNCERTAINTY

by

Tianyi Wang

Florida International University, 2015

Miami, Florida

Professor Gang Quan, Major Professor

Real-time systems are computing systems that demand the assurance of not only the logical correctness of computational results but also the timing of these results. To ensure timing constraints, traditional real-time system designs usually adopt a worst-case based deterministic approach. However, such an approach is becoming out of sync with the continuous evolution of IC technology and increased complexity of real-time applications. As IC technology continues to evolve into the deep sub-micron domain, process variation causes processor performance to vary from die to die, chip to chip, and even core to core. The extensive resource sharing on multi-core platforms also significantly increases the uncertainty when executing real-time tasks. The traditional approach can only lead to extremely pessimistic, and thus, unpractical design of real-time systems.

Our research seeks to address the uncertainty problem when designing real-time systems on multi-core platforms. We first attacked the uncertainty problem caused by process variation. We proposed a virtualization framework and developed techniques to optimize the system's performance under process variation. We further studied the problem on peak temperature minimization for real-time applications on multi-core platforms. Three heuristics were developed to reduce the peak temperature for real-time systems. Next, we sought to address the uncertainty problem

in real-time task execution times by developing statistical real-time scheduling techniques. We studied the problem of fixed-priority real-time scheduling of implicit periodic tasks with probabilistic execution times on multi-core platforms. We further extended our research for tasks with explicit deadlines. We introduced the concept of harmonic to a more general task set, i.e. tasks with explicit deadlines, and developed new task partitioning techniques. Throughout our research, we have conducted extensive simulations to study the effectiveness and efficiency of our developed techniques.

The increasing process variation and the ever-increasing scale and complexity of real-time systems both demand a paradigm shift in the design of real-time applications. Effectively dealing with the uncertainty in design of real-time applications is a challenging but also critical problem. Our research is such an effort in this endeavor, and we conclude this dissertation with discussions of potential future work.

TABLE OF CONTENTS

CHAPTER	PAGE
1. INTRODUCTION	1
1.1 Real-Time Systems	1
1.2 The Shift From Single Core To Multi-Core Design	4
1.3 Real-Time Multi-Core Systems Under Uncertainty	8
1.3.1 Uncertainty In Computing Infrastructures	8
1.3.2 Uncertainty In Real-Time System Designs	13
1.4 The Research Problem And Our Contributions	15
1.5 Structure Of The Dissertation	17
2. BACKGROUND AND RELATED WORK	19
2.1 Modeling Of Real-Time Systems	19
2.1.1 The Behavior Model	19
2.1.2 The Architecture Model	21
2.1.3 Real-Time Scheduling Policies	22
2.2 Related Work On Real-Time Scheduling	26
2.2.1 Single Core Scheduling	26
2.2.2 Multi-Core Scheduling	27
2.3 Related Work On Uncertainty	32
2.3.1 Related Work On Process Variation	32
2.3.2 Related Work On Statistical Response Time Analysis	34
2.4 Summary	37
3. TOPOLOGY VIRTUALIZATION FOR THROUGHPUT MAXIMIZATION ON MANY-CORE PLATFORMS	38
3.1 Related Work	38
3.2 Preliminary	42
3.2.1 Motivating Example	43
3.2.2 System Models	45
3.2.3 Problem Formulation	47
3.3 Our Approach	47
3.3.1 A Simple Workload/Performance Matching Heuristic	48
3.3.2 Opportunity Cost Based Workload/Performance Mapping	49
3.3.3 Logical/Physical Topology Mapping With Communication Awareness	52
3.4 Experimental Results	53
3.4.1 Experimental Setup	54
3.4.2 Performance vs. Mesh Sizes And Group Numbers	56
3.4.3 Performance vs. Different Communication/Execution Ratios	57
3.4.4 Computational Cost	58
3.5 Summary	59

4. HETEROGENEITY EXPLORATION FOR PEAK TEMPERATURE REDUCTION ON MULTI-CORE PLATFORMS	61
4.1 Related Work	61
4.2 Preliminary	65
4.2.1 System Models	65
4.2.2 Power And Thermal Models	66
4.2.3 Problem Formulation	68
4.3 Temperature Dynamics Formulation	68
4.4 Physical To Logical Mapping Heuristics	71
4.4.1 A Simple Utilization/Leakage Matching Heuristic	72
4.4.2 Hot-Cold Swapping	73
4.4.3 Enhanced Hot-Cold Swapping	75
4.5 Experimental Results	75
4.5.1 Experimental Setup	76
4.5.2 Temperature vs. Leakage Variation	77
4.5.3 Peak Temperature Minimization	79
4.5.4 Operational Costs	81
4.6 Summary	82
5. MULTI-CORE FIXED-PRIORITY SCHEDULING OF REAL-TIME TASKS WITH STATISTICAL DEADLINE GUARANTEE	84
5.1 Related Work	84
5.2 Preliminary	87
5.2.1 System Models And Problem Formulation	87
5.2.2 Motivations	88
5.3 Harmonic Index For Tasks With Probabilistic Execution Times	91
5.3.1 Mean-Based Harmonic Index	91
5.3.2 Variance-Based Harmonic Index	93
5.3.3 Harmonic Index Based on Cumulative Distribution Function	94
5.3.4 The Utilization Sum Based Harmonic Index	96
5.4 Task Partitioning Algorithm	97
5.5 Experimental Results	98
5.5.1 Experiment Setup	99
5.5.2 Performance w.r.t. Number Of Cores	99
5.5.3 Performance w.r.t. Schedulability	100
5.5.4 Computational Costs	101
5.6 Summary	102
6. ON THE HARMONIC FIXED-PRIORITY REAL-TIME TASKS WITH EXPLICIT DEADLINES	105
6.1 Related Work	105
6.2 Preliminary	108
6.2.1 System Models And Problem Formulation	108

6.2.2	Motivations	110
6.3	Harmonic Tasks With Explicit Deadlines	111
6.4	Harmonic Index For Tasks With Explicit Deadlines	118
6.5	Task Partitioning Algorithms For Tasks With Explicit Deadlines	121
6.5.1	Greedy Intensity Maximization Algorithm	121
6.5.2	Harmonic-Aware Clique Maximization Algorithm	122
6.6	Experimental Results	124
6.6.1	Experiment Setup	125
6.6.2	Performance vs. System Utilizations	126
6.6.3	Performance vs. Number Of Tasks	127
6.6.4	Statistical Model Evaluation	133
6.7	Summary	134
7.	CONCLUSIONS AND FUTURE WORK	137
7.1	Summary	137
7.2	Future Work	138
	BIBLIOGRAPHY	141
	VITA	159

LIST OF TABLES

TABLE	PAGE
4.1 Experiment parameters	77
5.1 Sub harmonic task set transformations of a 4-task set	92
6.1 A task set with six tasks	110
6.2 A task set with three tasks.	112
6.3 Intensity changes for task τ_3	113

LIST OF FIGURES

FIGURE	PAGE
1.1 Embedded system market [156]	2
1.2 Time line of multi-core history [19].	5
1.3 Demand for multi-core based devices	6
1.4 Performance comparison between multi-core processors and single-core processor[60].	7
1.5 Circuit delay and leakage current affected by process variation [33].	11
1.6 Comparison between micro-architecture- and core-level redundancy [148, 177]	12
2.1 Execution times of X and Y have independent Gaussian distributions $N(\mu, \sigma^2)$. The WCET is calculated as $\mu + 3\sigma$. $X + Y$ follows $[N(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2)]$. Therefore, $WCET_{X+Y} < WCET_X + WCET_Y$. [162] .	35
3.1 A framework to take advantage of performance heterogeneity to optimize system performance. An advanced built-in-self-test module is associated with each chip to collect the performance characteristics of the chip. The collected information is then used to map the physical hardware architecture to the logical architecture based on which the nominal design is conducted, with the goal to maximize the performance of the nominal design on this particular chip.	39
3.2 A motivation example.	44
3.3 An illustrative example for opportunity cost and performance metric. The colors and shades of task nodes in (a) imply the corresponding assignment to the logical topology: $v_0 \rightarrow C_{0,0}^l$, $v_1 \rightarrow C_{1,0}^l$, $v_2 \rightarrow C_{0,1}^l$, $v_3 \rightarrow C_{1,1}^l$	49
3.4 Performance vs. different mesh sizes and different group numbers	54
3.5 Performance vs. different communication/execution ratios	55
3.6 Computational Time comparisons for different algorithms on 10×11 mesh	58
3.7 Computational Time and Improvement comparisons for different iterations on 10×11 mesh	59

4.1	The topology virtualization framework. (a) Target application designed based on the nominal parameters of a 3×3 logical topology; (b) Configure the practical topology of an individual processor differently to mirror the logical topology and optimize the performance of the target application; (3) The physical topology, a 3×4 mesh, for the processor.	63
4.2	Voltage-Frequency variation example between cores.	65
4.3	Temperature comparisons between different utilizations	78
4.4	Peak temperature reductions normalized to initial mapping	79
4.5	Temperature reductions between <i>EHCS</i> and Exhaustive search	81
4.6	Computation time differences between different approaches	82
5.1	Processor usage versus task number with $DMP_{\Gamma} = 10\%$	101
5.2	Feasible mapping percentages for different approaches	102
5.3	Computational costs of approaches with different harmonic indexes with $DMP_{\Gamma} = 10\%$	103
6.1	Intensity varies with different deadlines.	113
6.2	Performance vs. system utilizations	128
6.3	Performance vs. number of tasks on 4 processors	129
6.4	Performance vs. number of tasks on 8 processors	130
6.5	Performance vs. number of tasks on 12 processors	131
6.6	Deadline miss probability vs. number of tasks.	135
7.1	Dark silicon trends for different technology nodes [144].	139

CHAPTER 1

INTRODUCTION

Real-time systems are the computing systems that not only need to deliver logically correct results but also deliver these results in a timely manner. A late result can be as bad as a wrong result. To function correctly, such a system needs high assurance to meet the required timing constraints under different conditions and application scenarios. Therefore, the worst-case based deterministic approach has been the common approach in the design of real-time applications. However, as real-time systems grow rapidly in both scale and complexity, and as IC technology marches into the deep sub-micron domain, the uncertainty has increased significantly to the degree that the traditional worst-case based design methodology becomes impossible or extremely pessimistic and impractical.

We seek to develop design techniques and methods that can efficiently and effectively deal with the uncertainty in the design of real-time systems. In this chapter, we first introduce the fundamentals of real-time systems. Then we discuss the existing uncertainty problems and opportunities when designing real-time systems on multi-core platforms. Next, we define our problems and our contributions. Finally, we present the structure of the dissertation.

1.1 Real-Time Systems

Real-time systems are defined as the systems in which the correctness of the system depends not only on the logical result of the computation, but also on the time when the result is produced [152]. A reaction that occurs too late to meet the timing constraint will be useless or even result in severe consequences. Today, real-time computing plays a vital role in our society as an increasing number of complex systems rely, partially or completely, on computer control. Examples of such ap-

plications that require real-time computing include: automotive applications, flight control systems, robotics, space missions, etc. In many cases, real-time computers are embedded into systems to be controlled spanning from portable devices (e.g., cell phones, watches, cameras) to larger systems (e.g., missiles, satellite, airplanes).

The embedded system market was valued at 121 billion dollars in 2011, and is expected to reach 194 billion dollars by 2018 with a CAGR (compound annual growth rate) of 6.8% from 2012 to 2018. According to Embedded Market Study 2014 [156], real-time capability is integrated in 61% of embedded projects as shown in Figure 1.1.

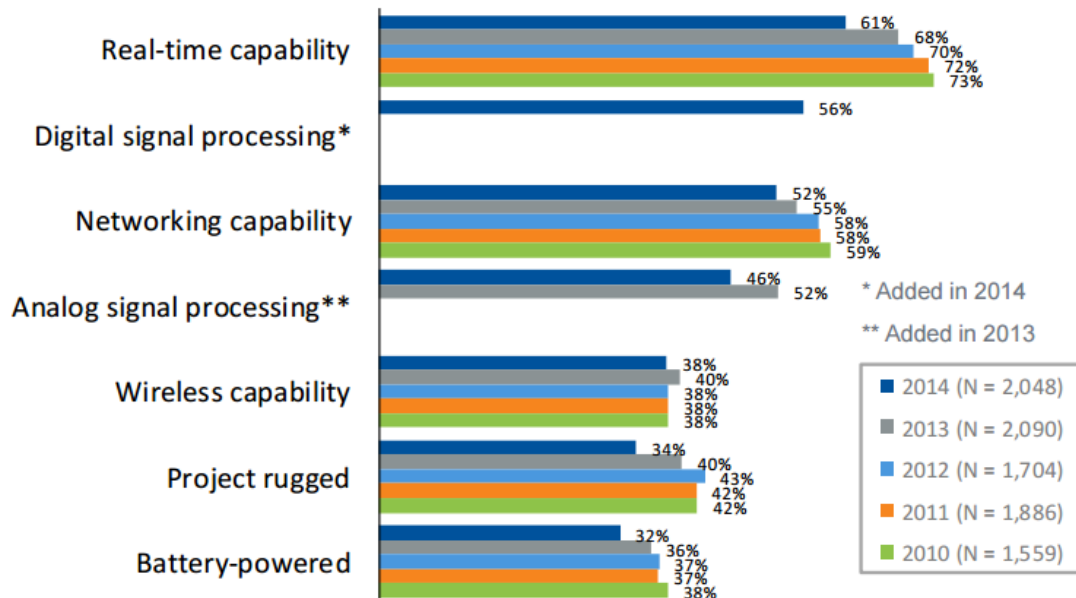


Figure 1.1: Embedded system market [156]

Compared to traditional computing systems, real-time systems must process information and produce a response within a specified time, (i.e., deadline), or else risk severe consequences. In general, real-time systems can be classified into two categories based on degree of timing correctness requirements: hard real-time systems and soft real-time systems. For hard real-time systems, it is required that

all the deadlines must be met for each instance, otherwise catastrophic consequences can occur if a deadline constraint is violated. For example, a real-time system that controls a nuclear power plant or aviation system cannot afford to miss deadlines of critical tasks since such a mishap could be catastrophic. More examples are mobile ABS systems, undersea exploration and space stations. On the other hand, soft real-time systems can tolerate a few or some deadline misses. Failure to meet deadlines can bring in degradation of certain degree of quality of service (QoS) but not catastrophic. Online streaming serves as an example. If some of video frames fail to decode or encode before deadlines, it will affect the video quality but will have no severe consequences. More examples are sound systems, MPEG players and internet telephones.

Unlike other traditional systems that have a separation between timing correctness and performance, real-time systems try to make a compromise between the two where timing correctness and performance are tightly coupled. One common misconception for real-time systems is that real-time computing must be “fast”. The goal of fast computing is to minimize the average response time or maximize the throughput of computing workloads. However, the goal of real-time scheduling is to meet the individual deadline of each task.

Just as ensuring deadline requirements is critical in real-time system design, another key concern of real-time system design is its predictability, i.e., how predictable the timing behavior of real-time tasks exhibit and to what degree they can satisfy deadline requirements of the system [152]. Fast or high performance computing helps to accelerate the computation of real-time tasks, but does not guarantee that all tasks can meet their deadlines.

As real-time systems become more and more complicated, real-time scheduling plays a key role to ensure deadline requirements in design of real-time systems, espe-

cially the systems with stringent deadlines. Real-time scheduling determines when and how to execute real-time tasks and utilize available resources most effectively such that tasks can be completed within specified deadlines. In addition, real-time scheduling is also a critical technology to make design trade offs between timing constraints and other system constraints, such as power/energy, thermal, reliability, etc. As a result, efficient and effective scheduling techniques are vital to solve problems targeting real-time systems.

Real-time scheduling can be categorized along different dimensions [130, 38], such as static/dynamic scheduling, priority/non-priority based scheduling, single processor/multi-processor scheduling. While significant amount of research efforts have been conducted based on single processor platforms for the past few decades, many more researches recently are focused on multi-core platforms instead of the single-core platforms, echoing the recent paradigm shift in the computing industry.

1.2 The Shift From Single Core To Multi-Core Design

As Gordon Moore predicted in 1965, the number of transistors on a chip doubled every 18 to 24 months, have been the driving force behind the integrated circuit industry [142]. Starting with 0.8 um technology scaling in the early 90's, the transistor's feature size is reduced by a factor of 0.7 approximately every 24 months (2 years). For example, from Intel Pentium processor to Intel Pentium IV processor through mid 2003, the clock frequency was doubled every 18 to 24 months. However, this exponential trend of performance improvement for single core has come to an end due to its excessive dynamic power dissipation and design complexity (the maximum frequency for single core design is around 4Ghz). Static power is also increased along with dynamic power due to transistors' source to drain leakage

along with gate leakage. This leakage has exponentially increased with technology scaling but it was not until recently it became a significant portion of the overall power budget.

When considering the limitations associated with single core design where increasing clock frequency is prohibitive, companies and researchers were trying to find an alternative to the single core paradigm. Multi-core was, therefore, the natural evolutionary step to keep up with the ever-increasing performance. Figure 1.2 shows the development of processors. The ever-increasing frequency with single core design has levelled off since 2005. From then, the trend was shifted from increasing clock-rate to adding more number of cores on a single chip to compensate power budget and thermal issue accompanied with single core design.

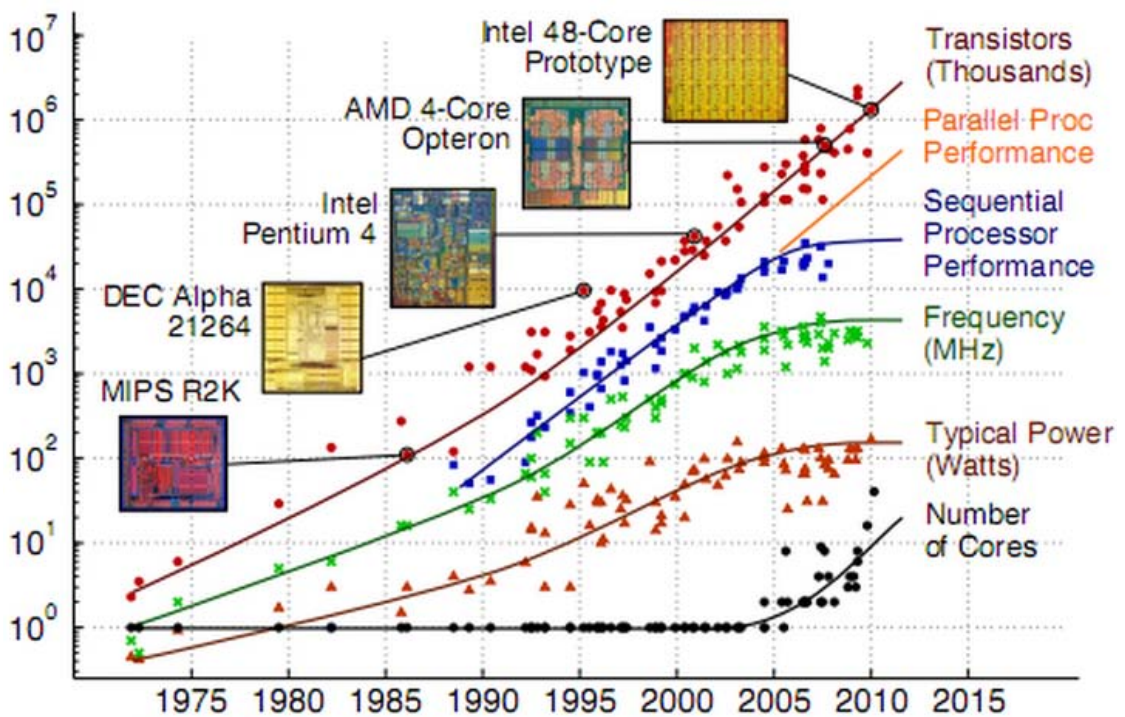


Figure 1.2: Time line of multi-core history [19].

With the continuous scaling down of the transistor’s feature size, billions of transistors are integrated on a single chip [79]. Multi-core architecture is becoming mainstream. Most desktop computers and server computers consist of multi-core or many-core high performance processors. For example, Intel has announced more advanced multi-core platforms that have 48 and 80 general purpose processors [76, 161]. Moreover, according to a research survey by IHS Inc. [1], it is forecasted in the research that starting from 2012, the expected shipment of multi-core processors will increase by 40% annually as shown in Figure 1.3. As a result, the design trend from single-core real-time systems to multi-core real-time systems is inevitable.

Demand for multi-core processing based devices (Source: IHS iSuppli)

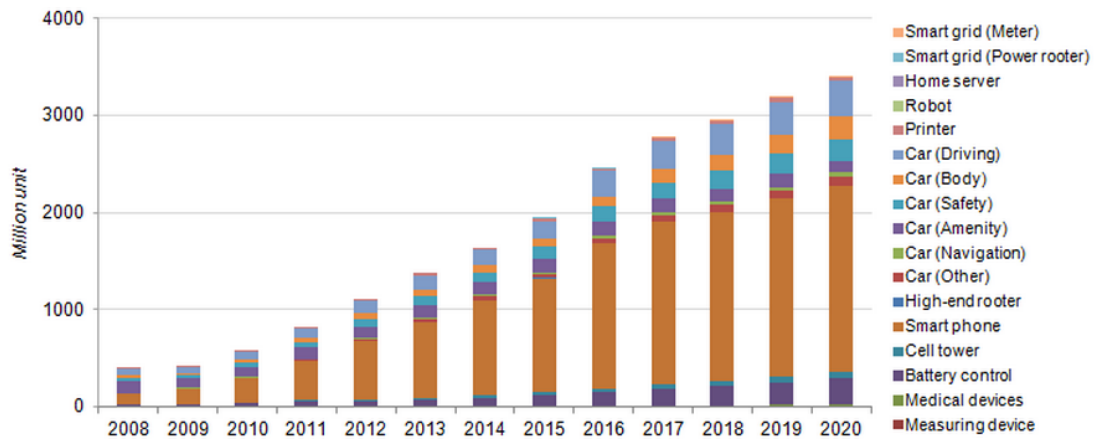


Figure 1.3: Demand for multi-core based devices

Multi-core design offers a number of unique advantages over the traditional single-core design. One of the advantages is that multi-core can exploit parallelism, which is one of the most effective ways to address the power issue, while maintaining high performance with lower voltage and frequency. Figure 1.4 shows the speedup of multi-core processors compared to a single-core processor. For example, a dual-core chip running multiple applications is about 1.5 times faster than a comparable single-core chip. Moreover, since the cores in a typical multi-core chip are on the

same die, they can share architectural components, such as memory elements. They thus have smaller chip area and lower costs than systems running multiple single core chips [60]. Also by integrating multiple cores on a single chip, communication latency can be reduced on the interconnects among cores and can achieve higher bandwidth than single core design. Another advantage with multi-core design is that multi-core design can provide redundancy compared to single core design, thus enhancing the resilience of the underlying architecture. Finally, time to market is a crucial factor in this industry. Multi-core design can facilitate IP re-use of cores from an SoC perspective and reduce the overall design effort [131].

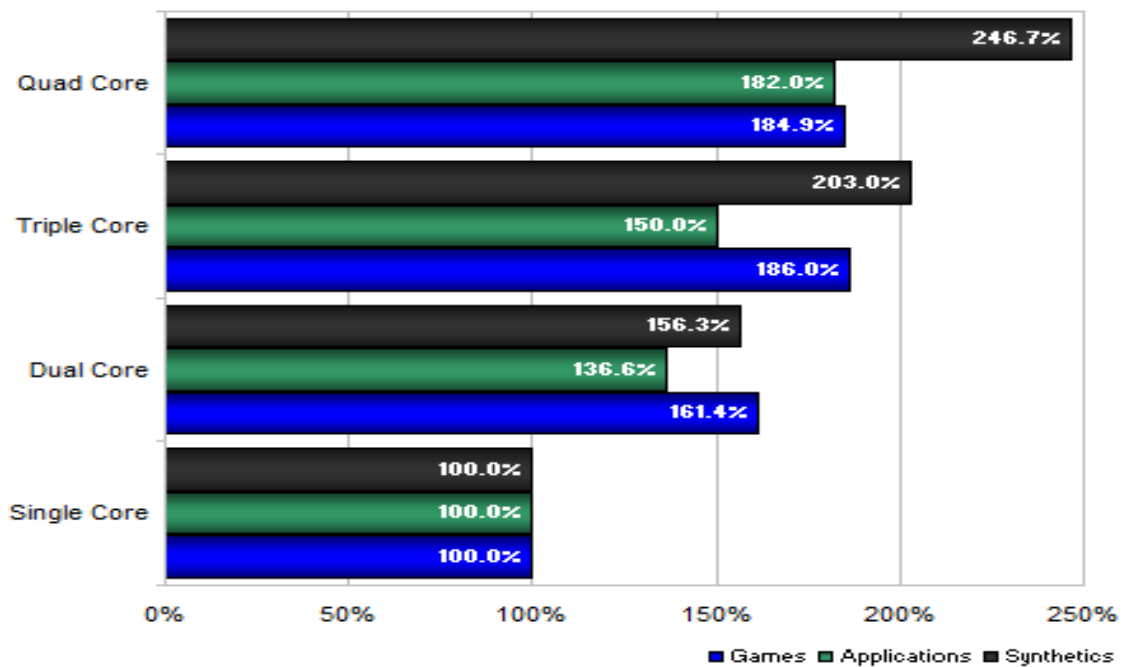


Figure 1.4: Performance comparison between multi-core processors and single-core processor[60].

With the shift from single core to multi-core design come along new challenges. One such challenge is directly dependent on how to efficiently utilize the hardware resource on multi-core platforms. Therefore, software plays a more important role in the multi-core era. Johnson et al. has stated that in the multi-core paradigm,

the complexity of scheduling on multi-core increases exponentially with the number of cores on a chip [83]. However, if software applications cannot fully utilize the hardware resources, the advantage with multi-core design will be diminished. It is well known that scheduling on multi-core platforms is an NP-hard problem [75]. Developing efficient and effective scheduling heuristics has been a common approach to the design of real-time systems. In the meantime, the uncertainty of real-time multi-core systems further complicates the problem.

1.3 Real-Time Multi-Core Systems Under Uncertainty

The uncertainty of real-time multi-core systems is two-fold. On one hand, as technology advances, it becomes much more difficult to precisely control the manufacturing process. As a result, the deviation between identical devices cannot be ignored for the underlying computing infrastructure [94] (for example, up to 30% frequency variation in 180nm technology [24]). On the other hand, real-time designs on multi-core systems are hardly deterministic, even under perfect technology scaling. One reason is that the smaller size the transistors are, the more vulnerable they are to environment changes (for example, 10% variation in dynamic power and 14x variation in leakage power for a 40°C change in temperature [168]), and this contributes to completion variation. Moreover, extensive resource sharing on multi-core platforms makes real-time system designs more complicated and indeterministic. In what follows, we discuss the aforementioned uncertainties in detail.

1.3.1 Uncertainty In Computing Infrastructures

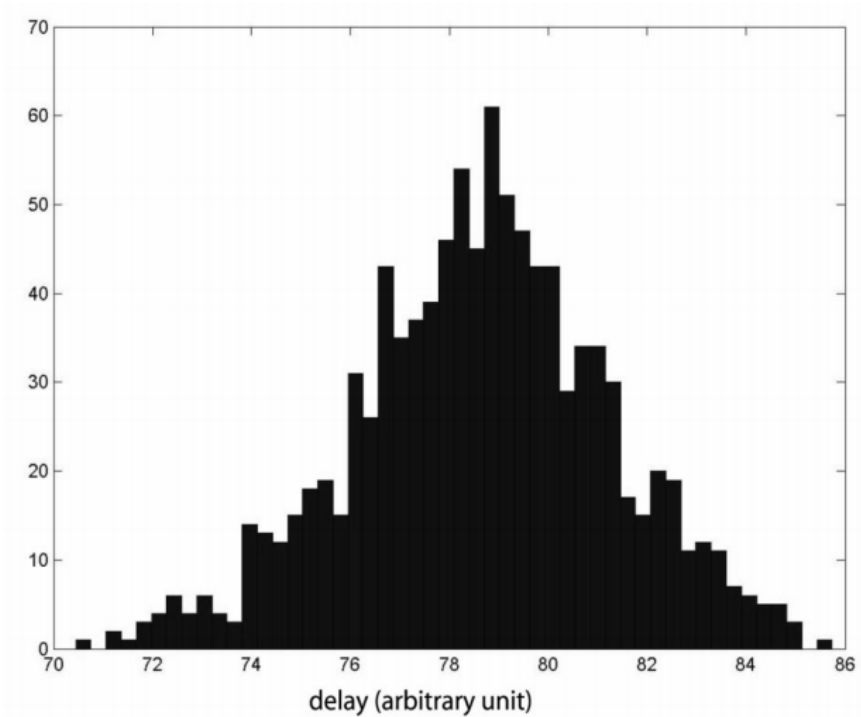
The first uncertainty of real-time multi-core systems is from manufacturing process and it is mainly caused by process variation. Process variation is the naturally occur-

ring variation in the attributes of transistors (length, width, oxide thickness) when integrated circuits are fabricated [94]. With the fast pace of technology scaling, the performance of IC chips becomes less and less deterministic. Process variations change the performance of IC chips (e.g., maximum clock frequency, power consumption, and etc). Frequency variation can be as much as 30% and up to 20x variation in chip leakage power consumption for a processor designed in 180nm technology [24]. Such indeterminism can significantly degrade the predictability of computing systems, which is critical for real-time systems. As a result, the uncertainty caused by process variation, not only impacts performance but also other design objective optimizations as well, such as power/energy consumption, and thermal issues. Figure 1.5 shows the circuit delay and leakage current histograms of the combinational logic circuit in a 0.1- μm CMOS technology due to transistor's gate length and threshold voltage variations. In Figure 1.5(a), the standard deviation of the path delay is more than 3.3% of the mean path delay, and in Figure 1.5(b), the standard deviation of the total leakage current is more than 9% of its mean, causing significant deviation of circuit performance and leakage power consumption [33].

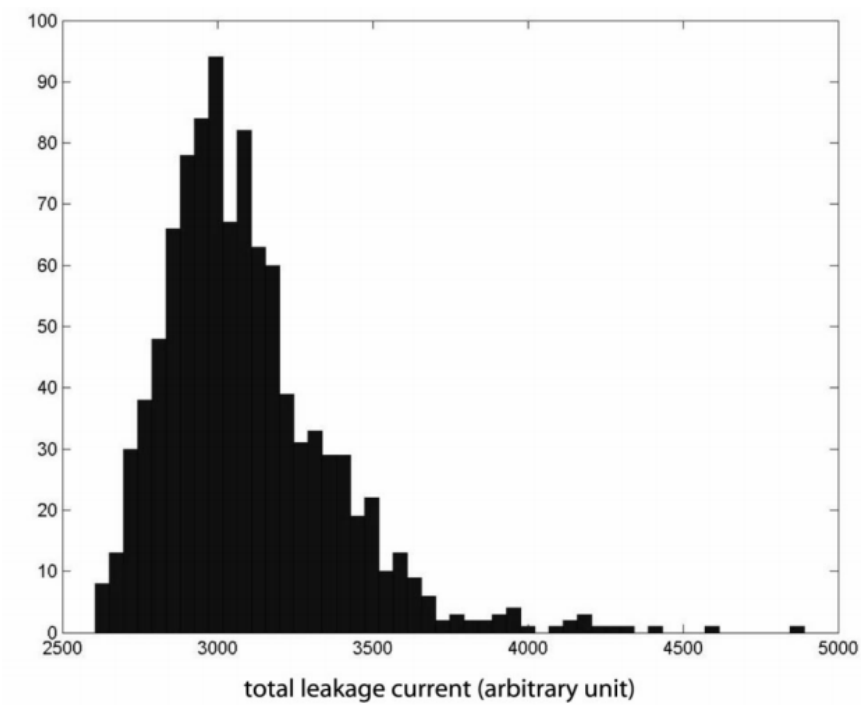
One major problem caused by process variation is the fabrication yield. Reduced feature size and increased chip area have increased the number and density of transistors on a single die, leading to a significantly decreased fabrication yield. According to [151], without considering defect tolerance during the architecture design phase, even under the best case, the yield of cell processors can be as low as only 10% to 20%. By adding spare cores on a chip, the yield rate can be improved to over 90%, according to [148]. Micro-architecture level and core level redundancies are the most popular strategies used in the industry where micro-architecture level redundancy refers to intra-processor redundancy and core-level redundancy refers to inter-processor redundancy. As shown in Figure 1.6(a), there is a crossover

point, such that the micro-architecture level has the same yield rate as core-level redundancy and from that point on, core-level redundancy constantly outperforms micro-architecture level redundancy. Similarly, in Figure 1.6(b), as technology keeps advancing, the yield rate becomes smaller and smaller if no redundancy mechanism is applied. At the same time, micro-architecture level redundancy brings better performance over core-level redundancy at more advanced technology generation because the chip has fewer cores and each faulty core can disable a large portion of the chip for core-level redundancy.

Another serious problem caused by process variation is performance variation, such as maximum clock frequency, leakage power dissipation, etc. It has been shown that the frequency variation can be as much as 30% and up to 20x variation in chip leakage power for a processor designed in 180nm technology [24]. Based on a test structure fabricated in IBM's 65 nm Silicon-On-Insulator (SOI) technology, Aarestad et al. [3] showed that worst case delay variations caused by chip-to-chip process variation can be as large as 21%. There are a few strategies that are commonly adopted to combat the process variation problems. For example, performance (or speed) binning approach, which intends to pack processors into different classes based on their maximum operating frequencies, is a common method for profit maximization in the presence of frequency variation [37, 140]. Performance binning is good for single core chips, however, it cannot capture the characteristic of a multi-core system. Since process variation may affect different cores differently, performance binning is much less effective to identify the performance for a multi-core system. Moreover, different applications may perform differently on the same multi-core system. For example, a multi-programmed application can achieve its maximum performance when the total performance of a multi-core system is maximized and a multi-threaded application can achieve its maximum performance when the performance of the lowest core of a



(a) Delay histogram of the combinational logic circuit.



(b) Histogram of the total leakage current of the combinational logic.

Figure 1.5: Circuit delay and leakage current affected by process variation [33].

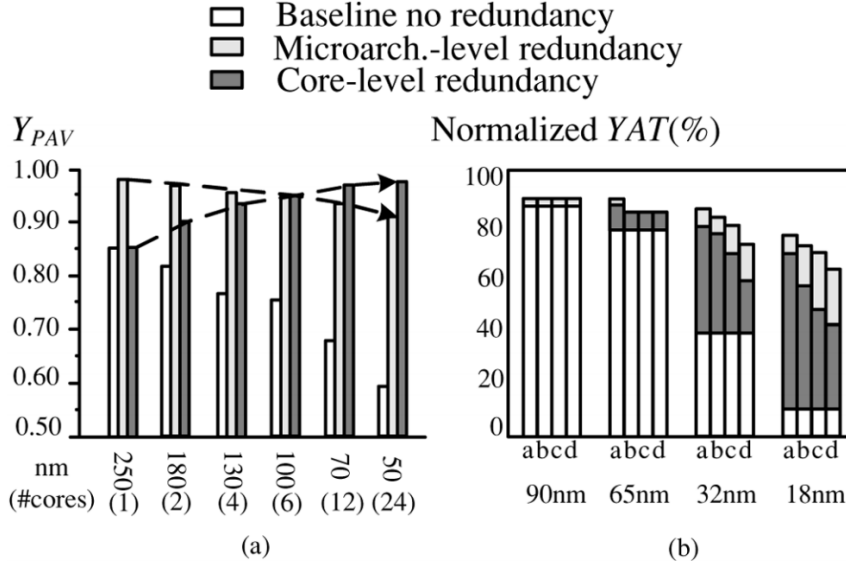


Figure 1.6: Comparison between micro-architecture- and core-level redundancy [148, 177]

multi-core system is maximized. Meanwhile, adaptive body biasing is another popularly adopted approach that can help to mitigate the increasing impact of process variation on leakage power dissipation which intends to apply a body-bias voltage to each die to reduce the leakage current and therefore, reduce the leakage power consumption. The disadvantage is that it can only deal with die-to-die variation, not with-die variation because all cores on the same die receive the same amount of body-bias voltage. To overcome this limitation, “body-bias islands” is proposed to solve with-die variations. The idea is to divide the entire die to different body-bias islands and each with its individual body-bias voltage to compensate leakage variation on each island. The effectiveness is depending on partitioning the die area into a number of “body-bias islands”. The more islands that are partitioned, the better improvement that can be achieved. However, the computational complexity also increases exponentially [56, 99, 57]. In the extreme case, one core forms one island, and this may lead to unacceptable circuitry cost.

The process variation problem exacerbates the uncertainty problem in real-time system design. As design parameters of processing cores deviate from their nominal values, the system design objectives can be severely compromised, or even worse, a computing system can malfunction or even fail catastrophically.

1.3.2 Uncertainty In Real-Time System Designs

The second uncertainty comes from the designs of real-time multi-core systems. As mentioned before, it is highly desirable that a real-time system has a high predictability to ensure a variety of timing requirements for different applications. Unfortunately, the current technology trends, from the perspective of either IC technology scaling or the paradigm shift to multi-core platform, are at odds with this design objective.

First, the continued IC technology scaling has greatly increased the variation in the performance of real-time system designs, e.g, power/energy consumption. The growing process variation causes computing performance and attributes to change from chip to chip and core to core, and has significantly widened the response time variance for real-time programs. The smaller transistor feature size also makes the processor sensitive to the changes in the operational environment. For example, when temperature changes from $20^{\circ}C$ to $60^{\circ}C$, as much as 10% variation in dynamic power and 14x variation in leakage power are measured for an ARM Cortex M3 processor [168].

Second, the resource sharing on multi-core platforms further exacerbates the problem, most notably, the execution time variation. In many computer systems, tasks share a processor and other resources such as data structures, sensors, etc. and they must operate on such resources in a mutually exclusive manner. Even on

a single processor, resource sharing can heavily impact timing behavior. For example, the NASA mission, Mars Pathfinder, nearly failed because of a resource-sharing protocol problem in the operating system [85]. Resource sharing on a multi-core platform is more complex because processors typically share low-level hardware resources such as caches, memory, and interconnects which make task execution times interdependent and therefore unpredictable [136]. A 30% slowdown for two tasks that shared the memory bandwidth was observed on a Pentium D processor [134]. Dasari et al. [36] proposed a method to bound the response time of tasks in a multi-core platform considering contention on the shared bus. Andersson et al. [8] presented a competitive algorithm for scheduling sporadic tasks on a multi-core platform with the assumption that a task may request one of the shared resources. They further extended their work to heterogeneous multi-core model, to schedule sporadic tasks on a t-type heterogeneous multi-core platform where tasks may share multiple resources [11].

To bound the execution requirement of real-time tasks has long been a challenging problem, since real-time programs can have very complicated structure and their execution times can vary significantly with different inputs. Traditionally, response analysis based on the worst case scenarios has been extensively explored, yielding to a certain degree of pessimism. Unfortunately, not all real-time systems can afford such pessimism and resource over-provisioning today. This is particularly true today as real-time applications expands in both scale and complexity, coupled with factors of continuous IC technology scaling and the design paradigm shift from single core to multi-core platforms.

The ineffectiveness of traditional deterministic approaches that are based on worst-case execution time calls for new real-time analysis and scheduling approaches to account for the uncertainty of response time analysis on multi-core platforms. The

statistical approach is just a promising approach. Instead of relying on worst-case execution time, statistical approach can construct the computation time of each task based on a probability density function, a mathematical expression approximating the real behaviour of the computation time [42]. In addition, the hard deadline guarantee may not be necessary for many soft real-time systems that allow a portion of the jobs miss their deadlines. For example, for aerospace industry, a probability of failure of 10^{-15} per hour is considered to be feasible compared to the maximum allowed probability of failure of 10^{-9} per hour that is required by the certification authorities [2]. Therefore, the statistical approach, takes system probabilistic characteristics, such as the execution times, into account to prevent over-provisioning and, at the same time, meet real-time constraints [43] which is a more favorable approach for real-time multi-core system analysis and design.

1.4 The Research Problem And Our Contributions

Our research in this dissertation focuses on addressing the uncertainty problems in the design of real-time systems on multi-core platforms. Specifically, we are interested in developing real-time scheduling techniques and analysis methods to ensure real-time constraints and at the same time, to optimize design criteria, such as performance maximization, peak temperature reduction, and energy minimization.

Toward this problem, we have made the following contributions:

1. First, we studied the problem on how to reduce the total schedule length of a task graph when realizing its nominal design on a Network-on-Chip(NoC) based many-core platform with faulty cores. We propose a framework called *topology virtualization* to deal with the faulty cores and process variation problem. Different from traditional approaches to re-define the mapping/scheduling

decisions in the nominal design, our methods judiciously mirror the physical architecture of each individual platform to the logical platform, based on which nominal design was conducted. To facilitate the physical/logic architecture virtualization, we developed a performance metric based on the *opportunity cost*, a concept borrowed from the economics field. Three virtualization heuristics were developed. Our experimental results show that the proposed approach could achieve up to 30% with an average 15% performance improvement by taking advantage of the heterogeneity of each individual platform.

2. Then, we targeted the problem on how to reduce the peak temperature of a real-time application by exploring our topology virtualization framework.. We developed three computationally efficient algorithms for deploying applications to individual devices. Our simulation study has clearly shown that, by taking advantage of the uniqueness of each individual physical chip, the proposed approaches can achieve $14.09^{\circ}C$ temperature reduction in average and less than $5^{\circ}C$ difference compared with *exhaustive search*. The experiments also show that these approaches are efficient and have low operational cost, 10^4 times faster than *exhaustive search*.
3. Next, we adopted a statistical approach to deal with the uncertainty for fixed-priority preemptive scheduling of real-time tasks on multi-core platforms with statistical performance guarantee. Rather than using a single-valued worst-case execution time (WCET), we formulated the task execution time as a probabilistic distribution. We developed a novel algorithm to partition real-time tasks on multiple homogenous cores, which takes not only task execution time distributions but also their period relationships into consideration. Our extensive experimental results show that our proposed methods can greatly improve the schedulability of real-time tasks when compared with traditional

bin packing approaches, 0.5 core savings for 8 tasks and 1.4 cores savings for 24 tasks.

4. Finally, we proposed a partitioned approach for fixed-priority preemptive scheduling of real-time tasks with explicit deadlines on multi-core platforms with timing constraint guarantees. We developed two partitioning heuristics based on a novel metric that can quantify the degree of harmonicity between two tasks. Our extensive experimental results show that our approaches can greatly improve the schedulability of real-time tasks when compared with existing works. Specifically, for 4-processor case, HCM in average can achieve around 12% and 8% improvement over DCT and FF, respectively. For 8-processor case, HCM in average can achieve around 15% and 12% improvement over DCT and FF, respectively. For 12-processor case, HCM has around 19% and 16% improvement over DCT and FF, respectively.

1.5 Structure Of The Dissertation

The rest of this dissertation is organized as follows. In Chapter 2, we introduce background to this dissertation and discuss related works that are close to our research problems. In Chapter 3, we study the problem on task scheduling on multi-core platforms with consideration of process variation. In Chapter 4, we target the problem on how to reduce the peak temperature of a real-time application on multi-core platforms with consideration of process variation. In Chapter 5, we propose a probabilistic approach for fixed-priority preemptive scheduling of real-time tasks on multi-core platforms with statistical deadline miss ratio guarantee. In Chapter 6, we present a partitioned approach for fixed-priority preemptive scheduling of real-time tasks with explicit deadlines on multi-core platforms with timing constraint

guarantees. Finally, in Chapter 7, we conclude this dissertation and discuss possible future work.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, we introduce the fundamentals on real-time systems and real-time scheduling technology. We then discuss the related research that deals with process variations and execution uncertainty.

2.1 Modeling Of Real-Time Systems

A real-time system is the one that processes information within a given specific timing constraint (i.e., deadline) to generate a corresponding response. If some tasks fail to finish before deadlines, the results that are produced maybe useless or they may lead to consequences that are catastrophic. In general, a real-time system consists of three components: the behavior model, the architecture model and the scheduling policy. In what follows, we introduce each part in details.

2.1.1 The Behavior Model

Behavior models are the models used to capture real-time tasks' characteristics. They can be denoted as applications, tasks, sub tasks, task nodes and agents to describe a real world application to be performed in reaching user's goal. They have shown to be very efficient and effective in designing, analysing and evaluating real-time applications.

In most real-time systems, a real-time task can be represented by a tuple: $\tau_i = \{c_i, r_i, d_i, p_i\}$, where c_i is the worst/average/best case execution time, r_i is the release jitter, d_i is the relative deadline and p_i is the period. A task can be periodic (if p_i is a fixed constant) [32], aperiodic (if p_i is a random variable that can be any value) [71] or sporadic (if p_i is the minimal inter arrival time between any two consecutive jobs

of that task) [18]. A real-time system can be soft, hard or firm depending on the quality of service requirements on tasks' deadlines. For a soft real-time system, even if the systems fails to meet the deadline (one or multiple times), the system is not considered to have failed (e.g., video streaming). For a hard real-time system, if the system fails to meet the deadline even once the system is considered to have failed (e.g., space stations). A firm real-time system is in between of soft real-time system and hard real-time system. Deadline missing is tolerable but not desired and missing deadline could cause potential loss of revenue (e.g., forecast systems) [93].

Besides the common parameters we have introduced above, inter-task dependency is another parameter that describes the characteristics of different task models. Therefore, tasks can be classified into independent tasks [165] and tasks with dependencies [167]. Independent tasks are those in which the execution of a task does not rely on the information of any other task, while tasks with dependencies can start execution only after all the precedent tasks have completed their executions. When dealing with task dependencies, researchers usually construct a directed acyclic graph (DAG) to represent a task model with dependencies [157]. The nodes and edges on a DAG normally have weights associated with them, denoting computation time to execute a task and the communication costs required between two different nodes, respectively. A classic heuristic to schedule a DAG is called Kernighan-Lin algorithm [89]. The algorithm attempts to partition the original DAG into two sub-graphs such that the overall weight of the edges between the two sub-graphs is minimized.

2.1.2 The Architecture Model

Architecture models are the models used to capture the physical characteristics of underlying computing infrastructure for real-time systems. In general, system architecture models can be classified into two categories: single-core model [113] and multi-core model [147]. Due to limitations of single-core design, researchers have moved their focus to multi-core design. Real-time scheduling on multi-core platforms is much more complicated than on single-core platforms since when we want to schedule a task, we need to decide not only when to execute the task but also where to execute it, which is well known as an NP-hard problem [75].

Multi-core platforms can be categorized into: homogeneous platforms and heterogeneous platforms. In a homogeneous platform, all the processors are identical. Thus, we don't need to store the detailed information for each processor except a list of tasks that have been already assigned on it. On the other hand, in a heterogeneous platform, each processor may be different from the other processors, certain tasks that have specific resource requirements can only be successfully assigned to a portion of processors while others are unable to schedule these tasks. Thus, each processor needs to provide its detailed list of available resources and each task must specify its resource requirement.

As more and more processors are integrated on a single chip, the arrangement of various types of processors draws more attention. System on chip (SoC) and Network on chip (NoC) become a favorable way to construct multi-core platforms [81]. SoC is an integrated circuit that integrates all components of a computer or other electronic system into a single chip. The basic composition of a typical SoC are micro-controller or digital signal processor (DSP) cores, memory blocks, peripherals, external interfaces, etc. NoC is a communication subsystem on an integrated circuit that connects between different intellectual property (IP) cores on a SoC chip. NoC

technology brings new methods to on-chip communication to improve communication delays between different processors compared with conventional bus-based and crossbar-based interconnects. It also improves the scalability and power efficiency of SoCs and therefore, NoC is the future trend of multi-core or even many-core design.

As we have mentioned above, on-chip communications play a significant role for multi-core platforms as communication delays consist a larger portion of total execution time between tasks than on single-core platforms. Moreover, the complex traffic patterns for on-chip communications also need to draw more attention since inefficient communication protocols may result in unbalanced traffics on a multi-core platform (e.g., the communication on one link maybe overloaded to create traffic jam while other links have few or no communication traffic at all) and may have deadlocks on communication links. There is extensive research on how to design an efficient and deadlock-free routing policies [61], such as X-Y routing [77] and odd-even routing [34].

2.1.3 Real-Time Scheduling Policies

Real-time scheduling policies are the design schemes that dictate when, where and how to run a real-time task on existing computing infrastructure. The key difference between a real-time scheduling and a non real-time scheduling (e.g., first in first out (FIFO) [13]) is that a real-time scheduling needs to guarantee timing. Given the behavior models and architecture models that are defined previously, a real-time scheduling policy determines where, when, and how to assign a set of tasks onto a multi-core platforms with the goal of optimizing design metrics (e.g, performance, temperature, power/energy consumption or reliability) and at the same time satisfying all real-time constraints, such as deadlines. Real-time scheduling

is a resource management scheme in nature and can be categorized from different perspectives. For the perspective of task behaviors, real-time scheduling can be classified into hard/soft. From the perspective of scheduling mechanisms, real-time scheduling can be classified into static/dynamic, priority/non-priority driven, preemption/non-preemption, etc. From the perspective of computing infrastructure, real-time scheduling can be classified into single core/multi-core. In what follows, we present a detailed discussion about real-time scheduling policies according to different categorizations we have introduced above.

Hard Real-Time vs. Soft Real-Time: Schedulers of real-time systems can be categorized as either hard real-time or soft real-time depending on task characteristics. Hard real-time means that all the jobs from each task have to be completed before their deadlines [30]. If the system fails to meet the deadline even once the system is considered to have failed. Many of these systems are safety critical and an overrun in response time leads to potential loss of financial damage, life or even catastrophe. Some examples are nuclear systems, medical applications such as pacemakers, a large number of defense applications, aviation, space station, etc. On the contrary, soft real-time does not necessarily require all the jobs to meet their deadlines (i.e., some deadline overruns are tolerable), but not desired [88]. There are no catastrophic consequences of missing one or more deadlines. However, there is a cost associated to overrunning which normally correlated with quality of service (QoS). A good example will be sound system in a computer. If a few bits are missed, nothing catastrophic will happen. But if more and more bits are missed, the performance of the sound system will degrade eventually.

Static vs. Dynamic: Depending on the time when the necessary information on completion of a task is available, real-time scheduling can be classified into static or dynamic. If real-time applications can be represented by tasks with known in-

formation such as execution times, relative deadlines and periods, scheduling can be done statically during compile time [87]. Static scheduling can take advantage of tasks with design parameters known as a priori to optimize the design goals. It is important to note that since static schedulers make all the scheduling decisions off-line, they can afford expensive techniques producing better results at the cost of high computational complexity. When the knowledge of task completion constraints is not available to the system, dynamic scheduling can be used. A dynamic scheduler uses the run-time information of the system resources to make decisions on the feasibility of executing real-time tasks [102]. In dynamic scheduling schemes, the priorities of tasks are assigned at run-time. Information about current and future availability of system resources are taken into consideration when making scheduling decisions. Based on such information, a dynamic scheduler can determine whether a new task can be schedulable, current tasks need to be dropped due to missing deadlines, or the priorities of tasks need to be re-evaluated. As a result, dynamic scheduling can provide greater system availability to make scheduling decisions since the priorities of tasks can be adjusted to the changes in the system environment. One major limitation of dynamic schedulers is the computational complexity as fast and efficient scheduling is the main concern because the computational costs of the schedulers should not conflict with the processing of real-time tasks.

Priority And Preemption: Priority defines the execution order of tasks in priority driven real-time scheduling. Additionally, priorities can be fixed where priorities are assigned statically to each task and do not change over time, such as rate monotonic scheduling (RMS) [113] or dynamic, where priorities may change at run-time, such as earliest deadline first (EDF) [171]. On the other hand, for non-priority driven real-time scheduling, decisions are made based on other criteria or policy, such as

weighted round-robin scheduling, each task τ_i is assigned a weight w_i , and each task τ_i will receive w_i consecutive time slices each round.

Preemption is a very helpful mechanism to support priority-based scheduling where preemption refers to allowing a higher priority task to preempt a lower priority task while it is being executed [68]. Such mechanism improves the flexibility of the scheduler to produce a viable scheduling solution. It is well known that scheduling policies with preemption can achieve better performance than those without preemption. For example, on a single core, preemptive EDF can achieve utilization bound of 1 while non-preemptive EDF cannot guarantee such a utilization bound. However, they are also more complicated and require more resources than non-preemptive scheduling policies. Moreover, the overhead of preemption such that to stall a lower priority task to allow a higher priority task to execute and resume the lower priority task's execution later needs to also be taken into careful consideration when designing such scheduling policies. Finally, poor design of preemptive scheduling policies can result in starvation of low priority tasks.

Single Core vs. Multi-Core: According to different underlying computing infrastructures, real-time scheduling can be categorized into single core scheduling [113] and multi-core scheduling [28]. One major difference between single core scheduling and multi-core scheduling is that, for multi-core case, a scheduling should decide not only when but also where a task is assigned and executed. It is well known that multi-core scheduling is a NP-hard problem, and therefore more complicated to deal with compared to single core scheduling.

2.2 Related Work On Real-Time Scheduling

In this section, we would like to discuss previous research and related works that have been conducted in the area of real-time systems in detail.

2.2.1 Single Core Scheduling

Real-time scheduling on single core has been conducted extensively. Three classic priority driven real-time scheduling algorithms are of great importance on single processor: RMS, DMS and EDF. They are fundamentals for single-core scheduling and also play a key role in implementing multi-core scheduling algorithms.

Rate Monotonic Scheduling (RMS): It is well known that RMS is among the most effective uni-processor real-time scheduling algorithms. It is one of the best representatives of fixed-priority preemptive scheduling algorithms on uni-processor. It is demonstrated by Liu and Layland [113] that RMS is the optimal scheduling policy among all fixed-priority scheduling algorithms, i.e., if a task set is schedulable, then RMS can successfully schedule that task set. Specifically, all the tasks are statically prioritized based on tasks' periods. The smaller the period is, the higher priority that task will have. They also formally proved that the utilization bound is about 0.7 for RMS (a feasible schedule by RMS can be found if the task set's utilization is less than or equal to 0.7).

In particular, if task periods are harmonic (i.e., all the tasks' periods are integer multiple of each other), the task set can be successfully scheduled according to RMS algorithm with the task set's utilization as large as 1 [100]. Harmonic periods are demanded in a wide spectrum of industrial real-time applications such as avionics, submarines, and robotics [27, 108, 146, 12] as well as control systems with nested

feedback loops [51]. Tasks with a harmonic relationship have a smaller hyperperiod [137, 25], which is a key concern for time-triggered embedded systems [92].

Deadline Monotonic Scheduling (DMS): RMS is optimal for real-time tasks with deadlines equal to periods (tasks with implicit deadlines). When a task's deadline is smaller than its period (tasks with explicit deadlines), RMS is not optimal anymore. Audsley et al. [15] proposed DMS policy to address the problem. Specifically, when a task's deadline is less than its period, all the tasks are prioritized according to their deadlines instead of periods. Then each task is scheduled statically based on their priorities and do not change over time. It is the optimal scheduling policy for tasks with explicit deadlines. Note that, RMS algorithm in fact is a special case of DMS algorithm.

Earliest Deadline First (EDF): EDF scheduling is proposed by Liu and Layland [113]. This real-time scheduling algorithm is based on uni-processor dynamic-priority preemptive scheme. It is an optimal scheduling policy among all dynamic priority scheduling algorithms. Specifically, all the tasks are dynamically prioritized during run-time, and EDF algorithm searches for the task that has the earliest deadline and executes that task with highest priority. Due to the characteristics of EDF, the utilization bound can achieve up to 1 which means as long as a task set's utilization is no greater than 1, EDF can return a feasible solution on uni-processor.

2.2.2 Multi-Core Scheduling

The development of appropriate scheduling algorithms for multi-core platforms is not a trivial problem at all. The reason is that not only uni-processor algorithms cannot be directly applicable but also some of the simple and effective approaches are counter-intuitive for multi-core scenarios. Mok et al. [125] showed that an algorithm

that is optimal for single processor is not optimal anymore for multiprocessor system. It is not surprising since the optimal scheduling for multi-core platforms is NP-hard [55, 64, 106, 107]. Efforts are needed to address the scheduling problems on multi-core platforms such that suboptimal results can be found instead of searching for optimal solutions [52].

2.2.2.1 Global Scheduling vs. Partitioned Scheduling vs. Semi-Partitioned Scheduling

Multi-core scheduling can be classified into three categories: global scheduling, partitioned scheduling, and semi-partitioned scheduling. While the former two approaches have their unique pros and cons, and none of them dominate the other in terms of schedulability [28], semi-partitioned scheduling can achieve better schedulability than both global and partitioned scheduling [48].

Global Scheduling: Global real-time scheduling puts all the tasks in a global queue and tasks are permitted to migrate from one processor to another for execution [5, 50]. Specifically, the majority of the previous work on global scheduling have been focusing on job-level migration, where jobs of different tasks may preempt on one processor and later resume on another processor and jobs from the same task may execute on the same processor or different processors.

Global scheduling normally has fewer context switches/preemptions since the global scheduler only preempts a task when there are no processors idle [9]. Also the laxity availability for tasks that execute less than their worst-case execution times can be used by all other tasks instead of the tasks that are assigned to the same processor with them. Similarly, if there is a task that overruns its worst-case execution time, it is less likely that deadline failure of the entire system will be encountered. Finally, global scheduling is more suitable for open systems, as there

is no need to run load balancing and/or task allocation algorithms when the set of tasks changes. Dhall et al. [40] addressed global scheduling of periodic tasks with implicit deadlines on m processors. They showed that the utilization bound for global EDF is $1 + \epsilon$, where ϵ is arbitrarily small. Andersson [6] presented a global static-priority scheduling for tasks with implicit deadlines and proved that its utilization bound is approximately 0.382.

Partitioned Scheduling: Partitioned real-time scheduling assigns each task to a dedicated processor and no task migrations are allowed during run-time [7, 46]. Partitioned scheduling has less overall impact on the entire system if a task overruns its worst-case execution time since the task can only affect other tasks assigned on the same processor. Also, as all the tasks are allocated to their dedicated processor, there is no penalty in terms of migration cost compared with global scheduling. For example, for global scheduling, a job that is preempted on one processor and later resumed on another can potentially result in additional communication costs and cache misses which is not the case for partitioned scheduling.

Furthermore, partitioned scheduling applies individual run-queue on each processor rather than adopting a single global queue. For large systems, the overheads of manipulating with only single global queue can be quite costly. Finally, once the tasks have been allocated to their dedicated processors by partitioned scheduling, knowledge on real-time scheduling for uni-processor can be readily applied. Andersson et al [10] proved that the utilization bound for multi-core partitioned approach with fixed-priority scheduling is 50% per core. Fan et al [45] proposed a method to improve the existing utilization bound and based on the enhanced utilization bound they developed a new partitioned approach to partition tasks for multi-core platforms.

Semi-Partitioned Scheduling: Semi-partitioned scheduling allocates most tasks to their dedicated processors same as partitioned scheduling while allowing some of tasks to be split into several segments and allocated to different processors [88, 101, 66, 47]. Normally, no more than $M - 1$ number of tasks are split, where M is the total number of processors. It is a combination of global scheduling and partitioned scheduling and therefore it can outperform the two theoretically [101, 66, 65]. Moreover, Zhang et al. showed that the overhead of the task migrations by a semi-partitioned scheduling approach can be relatively low and therefore has insignificant effects on the schedulability. Extensive research on semi-partitioned scheduling proved that the utilization bound can achieve much higher than either global or partitioned approaches. Lakshmanan et al. [66] showed a utilization bound of 65% and Guan et al. [67] pushed the utilization bound to 0.7%.

2.2.2.2 Real-time Scheduling With Timing And Other Design Objectives

With the increasing popularity of multi-core platforms, to optimize different design objectives at the same time other than to guarantee timing alone has drawn more and more attention from both academia and industry. Power consumption and thermal issue are the two top tier concerns due to rising performance demand on multi-core real-time systems. In what follows, we introduce the related works that have been conducted from the perspectives of power consumption and thermal issue.

Power/Energy consumption: Power density is the major cause that pushes the shift from single-core scheme to multi-core scheme and it is still a top tier concern on multi-core platforms as the transistor count continues to grow [17]. Early work on power-aware scheduling is basically focusing on how to minimize dynamic power because dynamic power contributes to the major portion of the total power

consumption. A list of works have been done on the convex relationship between the dynamic power and clock frequency/supply voltage to minimize the dynamic power and at the same time, meet the deadlines of all the tasks by reducing the clock frequency and supply voltage [102, 171]. When technology advances further into deep sub-micron domain, leakage power becomes a significant portion and cannot be ignored anymore. Extensive works have been published on the problem of power-aware multi-core real-time scheduling with consideration of leakage power consumption [72, 73, 115, 182, 31, 180, 173, 118, 78, 181, 32, 170].

Thermal Issue: With billions of transistors integrated on a single chip to further drive the pace of multi-core design, high peak temperature has increasingly become a critical issue in computer system design. High chip temperature not only increases packaging/cooling cost (estimated at 1-3 dollar per watt [150]) but also significantly degrades system performance and reliability. A $10^{\circ}C$ to $15^{\circ}C$ increase of operation temperature can reduce the lifetime of a chip by half [172, 82]. Moreover, high chip temperature dramatically increases leakage power dissipation. The leakage power dissipation of a chip can be tripled when temperature increases from $45^{\circ}C$ to $110^{\circ}C$ according to [29], which in turn will further elevate temperature. Temperature constraint is becoming the first-class design concern for digital CMOS ICs. Chantem et al. [30] proposed an MILP-based solution to minimize the peak temperature for task graphs. Ukhov *et al.* [160] proposed a peak temperature estimation method to keep track of temperature dynamics of a multi-core system until steady state. Kumar et. al. [98] presented a stop-n-go approach to reduce the peak temperature for tasks with data dependencies by distributing slack time between jobs with the goal of minimizing peak temperature and no make-span was violated.

2.3 Related Work On Uncertainty

In this section, we introduce existing research that has been conducted to account for real-time scheduling under uncertainty in detail. Specifically, we first discuss about related work proposed to deal with process variation, and then we present previous study on statistical real-time scheduling.

2.3.1 Related Work On Process Variation

Significant research has been made to address the problems raised by process variation from layout/device level, micro-architecture level, and system level. In what follows, we will introduce the related work on process variation in details.

Layout/Device Level Design: Random dopant fluctuation (RDF) is one major cause for threshold voltage variation, therefore, many approaches have been proposed to improve process techniques to minimize RDF effect, such as decreasing in channel doping and gate oxide thickness [153]. Historical scaling which reduces gate oxide thickness also suggests a continued improvement in the random variation coefficient. Moreover, the introduction of HiK+MG in 45nm technology helps with historical scaling to mitigate the impact of RDF [94]. They further introduced a powerful tool for assessing process variation by locating ring oscillators routinely in all product designs. The detailed ring-oscillator data can later be used to identify areas of concern for process variations and layout techniques can be applied to minimize variation effects. For example, lay out matched devices so that they have the same centroid or center of gravity [96]. Agarwal et al. [4] presented an overview of test structures for characterizing statistical variation of process parameters and discussed the significance of design and measurement results dedicated towards modeling process variation. Kim et al. [90] proposed a process compensat-

ing dynamic circuit technique to maintain the performance of dynamic circuits and reduce the variation in delay and robustness. A leakage current sensor design was also presented to accurately measure leakage variation. There are also works that add built-in sensors or redundant devices on a chip to mitigate process variation [97]. However, it becomes increasingly challenging as transistors' feature size scales close to or below 10 nm [154].

Micro-Architecture And System Level Design: Besides extensive works on layout and device level, researchers are more interested to address process variation from micro-architecture and system levels. For example, performance/speed binning technique has been widely used to cluster chips with similar performance (e.g., frequency, leakage power) [141, 122]. However, this approach didn't consider within-die variation. A number of approaches adopt statistical task model to deal with process variation. Wang et al. [164] presented a variation-aware task allocation and scheduling algorithm for multi-core platforms to mitigate the impact of process variation. A new design metric, called performance yield, was developed to guide the task allocation and scheduling procedure. Sarangi et al. [139] proposed a micro-architecture aware model to account for process variation. A framework was also developed to model timing errors. Then, with the combination of the variation model and the error model, detailed statistics of different process parameters and operating conditions can be produced to estimate timing error rates for pipeline stages to account for process variation. Micro-architecture redundancy mechanisms are widely adopted to mitigate process variation by adding spare cores or entire micro-architectures to compensate faulty units [175, 174, 176, 179]. Specifically, Zhang et al. [176] proposed a row rippling column stealing mapping scheme to re-map the underlying architecture to minimize on-chip communication cost. They further extended their work to incorporate frequency variation as well in [179]. Yue

et al. presented an application specific task allocation method with the goal of maintaining the similar performance in the presence of faulty cores. [175, 174]. Adaptive body biasing (ABB) is also a popular scheme to reduce the impact on process variation. Tschanz et al. [159] proposed to apply a single body bias value per die to allow each die to meet the given frequency and power constraints. Garg et al. [59, 56] also proposed a system-level variability mitigation framework by applying ABB technique. They partitioned a multi-core platform into body-bias islands and assigned body-bias voltages for each island post-fabrication. Depending on the granularity of number of islands, significant run-time improvements over Monte-Carlo based technique were achieved while providing similar leakage power reductions.

2.3.2 Related Work On Statistical Response Time Analysis

Traditional real-time scheduling approaches that consider the previous mentioned task model are deterministic in nature. The well-known periodic task model for real-time systems assumes a worst-case execution time for each task and may be too pessimistic in terms of performance [124]. Specifically, Figure 2.1 shows that the simple addition of the worst-case execution time of two tasks can result in pessimistic estimation of the overall worst-case execution time and may end up over-provisioning resources to guarantee real-time constraints [162]. Tasks may have different execution times on multi-core platforms due to resource sharing and therefore it is reasonable to address real-time scheduling on multi-core platforms with statistical approaches to accurately evaluate the real-time constraints to achieve various design goals.

The statistical approach takes system statistical characteristics, such as the execution times, into account for real-time system analysis and design to prevent over-

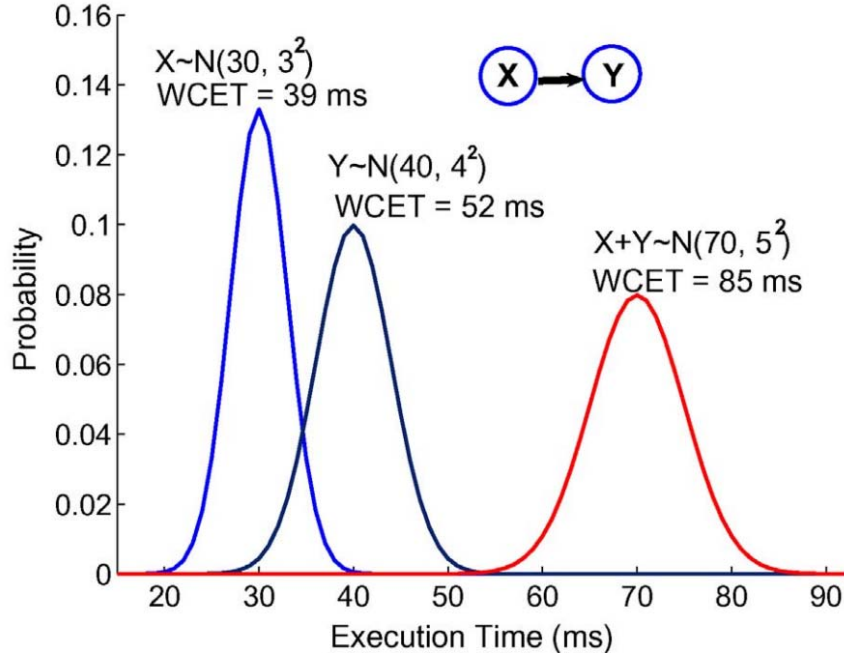


Figure 2.1: Execution times of X and Y have independent Gaussian distributions $N(\mu, \sigma^2)$. The WCET is calculated as $\mu + 3\sigma$. $X + Y$ follows $[N(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2)]$. Therefore, $WCET_{X+Y} < WCET_X + WCET_Y$. [162]

provisioning and, at the same time, meet real-time constraints [43]. There have been increasing interests from the real-time community on statistical approaches for real-time system analysis and design on a single core. For example, Tia et al. [158] presented a statistical performance guarantee for semi-periodic tasks by transforming semi-periodic tasks into a periodic task followed by a sporadic task. Atlas et al. [14] introduced a statistical rate monotonic scheduling for periodic tasks with statistical QoS requirements. Maxim et al. [119] proposed three priority assignment algorithms for statistical real-time systems. They further improved the previous work by proposing a framework of re-sampling mechanisms that can simplify the response time distributions in order to ease timing analysis for real-time systems in [121]. Yue et al. [117] presented a statistical response time analysis by analyzing samples in timing traces taken from real systems. In [91], the authors proposed a

stochastic analysis framework which computed the response time distribution and deadline miss probability for each individual task. The framework can be applied to both fixed-priority and dynamic-priority systems. The authors in [120] extended their work to allow both task's execution time and period to be random variables and computed analytically the response time distribution of the tasks under a task-level fixed-priority preemptive scheduling policy. In [16], the authors proposed a new convolution-based stochastic analysis in which they modeled faults as additional execution time to bound the probability to exceed a response-time value in the worst-case under fixed-priority non-preemptive scheduling policy.

On the other hand, the statistical approach on a single core can be readily applied to analyze the scheduling of tasks on a multi-core platform. Therefore, heuristics are needed to partition tasks with random execution times to better utilize the underlying resources. Goel et al. [62] studied the problem of makespan minimization when tasks are stochastic. They proposed stochastic load balancing methods such that the expected value of the maximum load on a processor is minimized. Wang et al. [162] proposed a task mapping method for tasks with probability distributions in a DAG on a homogeneous system. Specifically, Sum and Max functions were used to find the distribution of the partially scheduled task graph. Then a new task allocation algorithm was proposed to partition each task to its best candidate processor based on a metric called performance yield. Li et al. [110] also presented a task scheduling method for DAGs, where task processing times and communication times were random variables on a heterogeneous cluster system. A stochastic dynamic level scheduling algorithm was proposed, which was based on stochastic bottom levels and stochastic dynamic levels. Mills et al. [123] discussed the potential of global earliest deadline first (GEDF) scheduling algorithm for stochastic tasks on soft real-time multi-core systems. They proved that expected deadline tardiness

on multi-core platforms can be bounded under GEDF when task execution times are probabilistic. There are also some works that have been conducted to optimize power/energy, or temperature considering stochastic task models [169, 86, 109].

2.4 Summary

In this chapter, we introduce the fundamentals of our research and discuss about the related work from a variety of perspectives. We first give a brief introduction on real-time systems. Specifically, a real-time system contains three major parts: behavior model, architecture model and scheduling policy. Next, we discuss about the related works that have been conducted on real-time systems from single-core scheme to multi-core scheme. Then we present real-time scheduling from another perspective: statistical scheduling. Traditional deterministic approaches are not suitable for the future multi-core scheduling and there is a great need to study statistical real-time algorithms. Finally, we discuss real-time scheduling from the perspective of design objectives. Scheduling approaches based on two top-tier design objectives are presented, power consumption and thermal issue.

In this dissertation, our objective is to develop efficient and effective scheduling algorithms for real-time systems on multi-core platform, such that the design objectives can be optimized and at the same time, real-time constraints are satisfied. In the following four chapters, i.e. Chapter 3, 4, 5 and 6, we present our contributions. In Chapter 7, we conclude this dissertation.

CHAPTER 3

**TOPOLOGY VIRTUALIZATION FOR THROUGHPUT
MAXIMIZATION ON MANY-CORE PLATFORMS**

In this chapter, we first present our research on process variation-aware real-time scheduling on multi-core platforms to address uncertainty problems. Our goal is to minimize the execution latency of an application by properly mirroring a physical multi-core architecture, which may have faulty cores and significant core-level performance variations, to the logical architecture. Three virtualization heuristics are presented in this work. Specifically, we introduce a novel performance metric developed based on the *opportunity cost* [26], i.e. a concept originated from the economics domain, to guide our virtualization process. We have conducted extensive experimental studies to investigate the benefits of the proposed framework and heuristics. Our experimental results show that the heuristics can achieve up to 30% (with 15% in average) performance improvement (i.e. schedule length) over the existing methods.

3.1 Related Work

With the continuous scaling down of the transistor feature size, billions of transistors are integrated on a single chip [80]. Multi-core/many-core architecture is becoming mainstream. Most of desktop computers and server computers nowadays use high performance processors with multiple processing cores. Intel has announced more advanced many-core platforms consisting of 48 and 80 general purpose processing cores [76, 143, 161].

In the meantime, however, as transistor feature size continues to shrink to the degree below the wavelength of light used to print them, it becomes difficult to precisely control the manufacturing process. This can lead to significant variations

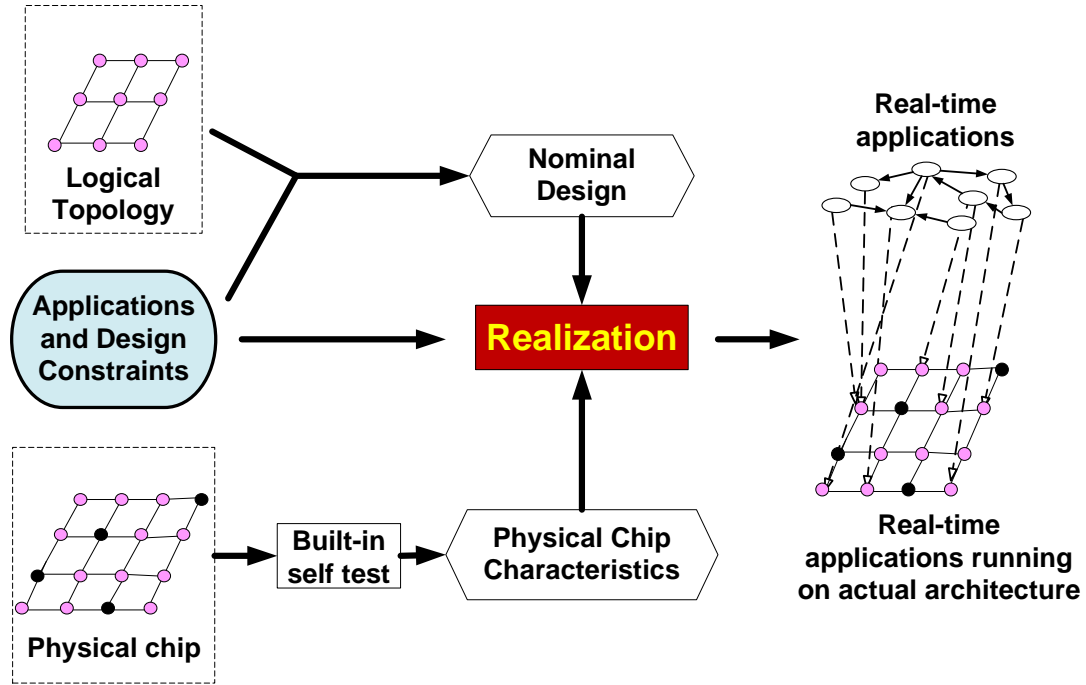


Figure 3.1: A framework to take advantage of performance heterogeneity to optimize system performance. An advanced built-in-self-test module is associated with each chip to collect the performance characteristics of the chip. The collected information is then used to map the physical hardware architecture to the logical architecture based on which the nominal design is conducted, with the goal to maximize the performance of the nominal design on this particular chip.

in key transistor parameters, such as transistor channel length, channel width, oxide thickness, and threshold voltage, which can further result in the maximal working frequency and power consumption of processing core varying from core to core and chip to chip [128, 94], even if all of them use the same, identical design. The 2008 International Technology Roadmap for Semiconductor (ITRS) [80] predicts that circuit variability will increase from 48% to 66% in the next ten years.

One major problem caused by manufacturing variations is the fabrication yield. Reduced feature size and increased chip area have increased the number and density of transistors on a single die, leading to a significantly decreased fabrication yield.

According to [151], without considering defect tolerance during the architecture design phase, even under the best case, the yield of cell processors can be as low as only 10% to 20%. Therefore, micro-architecture level and core-level redundancies are employed to improve the fabrication yield. According to [148], incorporating core-level redundancy, at or below 100 nm technology, will achieve better yield performance than micro-architecture level redundancy.

Another serious problem caused by manufacturing variations is performance variations, such as maximum clock frequency, power dissipation, etc. It has been shown that the frequency variation can be as much as 30% and up to 20x variations in chip leakage power for a processor designed in 180nm technology [24]. Based on a test structure fabricated in IBM's 65 nm Silicon-On-Insulator (SOI) technology, Aarestad et al. [3] showed that worst case delay variations caused by chip-to-chip process variations can be as large as 21%. As design parameters of processing cores deviate from their nominal values, the system design objectives can be severely compromised, or even worse, a computing system can malfunction or even fail.

Significant achievements have been made in recent research [94, 95] by employing new materials. However, layout techniques and other device technologies on mitigating performance variations, which are induced by manufacturing variations will become increasingly challenging as transistor size continues to scale towards dimensions close to or below 10 nm [154]. Besides extensive research on device and layout level techniques (e.g [94, 95]) to address manufacturing-variation problems, there are increasing interests to address this problem from architecture and system levels. For example, performance binning techniques are proposed (e.g. [141]) to cluster chips with similar performance. As a result, even in the presence of large manufacturing variations, processors of the same grade have less performance variations.

One drawback of this approach, though, is that it cannot deal with performance variations among different cores within the same chip.

In the presence of performance variations, it becomes too pessimistic to design a system based on worst-case scenarios. Therefore, instead of adopting the traditional design methodology which is based on deterministic parameters, several researchers incorporate statistical analysis into system-level design. Wang et al. [164] presented a task allocation and scheduling algorithm to map a task graph to a multi-core platform with the goal to maximize the performance yield rate, i.e. the probability that a processor can meet the desired performance of a given application. They further extended their work to consider not only performance differences of multiple cores, but also physical link differences in NoC as well. Momtazpour et al. [126] considered a similar task graph mapping problem on a multi-core NoC architecture, with the goal to maximize the percentage of manufactured chips that can meet power constraints for a given application. These statistical approaches try to optimize results in a probabilistic manner. However, from the perspective of an individual processor, the designs can be too optimistic or too pessimistic due to different performance variations.

We believe that the core heterogeneity due to performance variations, if handled properly, can in fact improve the performance of a nominal design. As a result, in this work, we are interested in developing appropriate *virtualization* techniques that can judiciously mirror physical architecture to logical architecture and at the same time improve the throughput of the nominal design on each individual hardware platform. Figure 3.1 illustrates the virtualization framework of our approach. We assume that each chip is equipped with an advanced *built-in-self-test*(BIST) module, that can detect faulty cores and capture performance variances when a device starts. Note that simple modules such as those introduced in [129, 94] can be easily incorporated

into a multi-core platform for detecting purpose. The performance characteristics captured by the BIST module will be used to mirror the logical architecture to the underlying physical architecture with the goal of maximizing the application’s performance.

A few studies [174, 175, 176, 178, 179] have been conducted which are closely related to our work. Zhang et al. [176, 178] proposed several heuristics to replace faulty cores with redundant cores to improve the fabrication yield. They further extended their work to deal with performance variations by constructing sub-meshes using cores with similar performance [179]. These approaches do not take application characteristics into consideration. A more recent work proposed by Yue et al. [174, 175] improved upon Zhang’s work [176, 178] by taking application characteristics into consideration, and intended to maintain the similar real-time performance after replacing faulty cores with redundant cores. These approaches only deal with faulty cores and do not take performance variations into consideration.

The rest of the chapter is organized as follows. In Section 3.2, we first use an example to motivate the research in this work and then formulate the problem. We discuss the virtualization heuristics we developed in Section 3.3. Experimental results are discussed in Section 3.4. We draw the conclusions in Section 3.5.

3.2 Preliminary

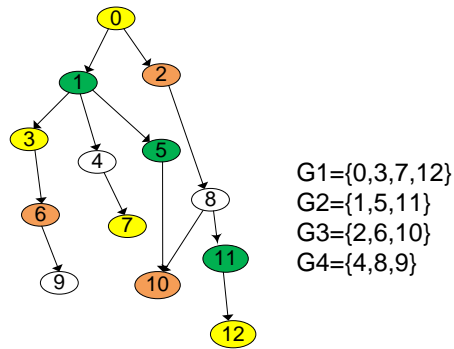
In this section, we first use a simple example to motivate our research. We then introduce the system models used in this work and define the research problem formally.

3.2.1 Motivating Example

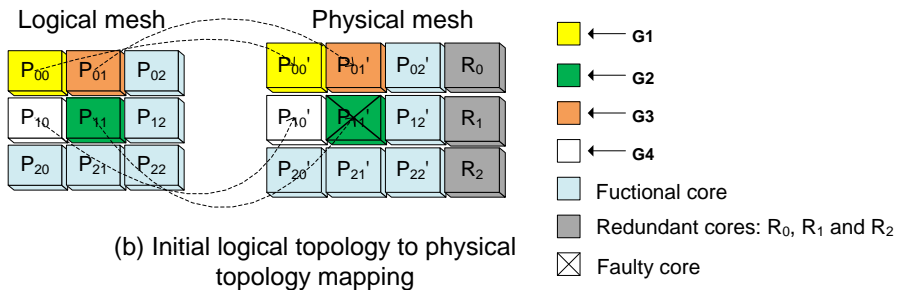
Consider an application where the task graph is as shown in Figure 3.2(a). The application is designed for a multi-core architecture with standard 3×3 mesh. Assume that the nominal design, i.e. the design based on the nominal performance of the chip, has been given and shown on the logical mesh in Figure 3.2(a): tasks with same colors and shades are assigned to the same core. For example, tasks 0,3,7, and 12 are mapped to core (0,0); tasks 2,6, and 10 to core (0,1); tasks 4,8, and 9 to core (1,0); and tasks 1,5, and 11 core (1,1).

Now consider when realizing this design on a practical platform. To improve the yield rate, manufacturers usually add redundant cores on the same chip. In our case, we assume the physical mesh size of the chip is 3×4 with 3 redundant cores shown in Figure 2(b). Assume that core (1,1) happens to be a faulty core. One approach is to replace the faulty core with a redundant core using the Row Rippling Column Stealing (RRCS) algorithm presented in [176] or similar approaches detailed in [174, 175]. However, these approaches do not take core-to-core performance variations into consideration. As shown in Figure 2(c), instead of replacing the faulty core only, we can re-map the physical architecture to the logical architecture to optimize the performance of the nominal design.

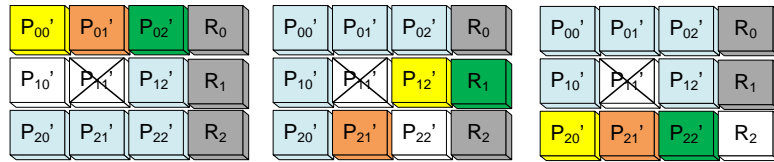
Since programmers make the nominal design solely based on the logical topology without being aware of what the physical topology really looks like, opportunities exist to mirror the logical topology based on the actual performance and other characteristics of the physical topology to optimize the system performance. For instance, in Figure 3.2(b), the logical mesh is 3×3 . When running application programs according to the nominal design, the operating system (OS) only cares about a logical mesh of 3×3 , without knowing how this logical mesh is mapped to the underlying physical mesh. As a result, we can judiciously choose the physical



(a) Task graph with group partitions



(b) Initial logical topology to physical topology mapping



(c) Possible physical mapping solutions

Figure 3.2: A motivation example.

topology to the logical topology mapping (such as Figure 3.2(c)) such that nominal design performance is maximized for each individual hardware platform.

Since we assume that we know the specifications of real-time applications, and using BIST module, we are able to know the exact performance for each core. Theoretically, we can then re-map and re-schedule task nodes accordingly. However, this essentially implies that we have to re-design the entire application for each individual platform, which can be expensive if not infeasible at all. Note that, by virtualizing each individual physical hardware architecture properly to the logical architecture, we have the potential to take advantage of the uniqueness of each chip to optimize the system's performance without the need to change its software.

3.2.2 System Models

In this section, we formally define our system models, including both application model and architecture model.

The *application* in this work is modeled as a directed acyclic task graph $G = \{V, E\}$. $V = \{v_1, v_2, \dots, v_k\}$ and each task node v_i represents a task in the application. We use $|v_i|$ to represent the execution time of task node v_i under the nominal frequency. $E = \{e(i, j) = (v_i, v_j) \mid \text{if task node } v_i \text{ communicates with task node } v_j\}$. Each arc, i.e. $e(i, j) \in E$ also indicates the dependency between two task nodes v_i and v_j with direction from task node v_i to task node v_j . A weight $w(e(i, j))$ is associated with each arch $e(i, j)$ to represent the data volume to be transferred from task v_i to v_j .

For the *logical architecture* (denoted as $A_{r \times c}^l$), we assume it consists of $r \times c$ homogeneous cores that form a standard $r \times c$ mesh architecture, i.e.

$$A_{r \times c}^l = \{C_{i,j}^l, i = 0, \dots, r - 1; j = 0, \dots, c - 1\}. \quad (3.1)$$

where $C_{i,j}^l$ represents the core located at position (i, j) in the logical mesh architecture. We assume that each core has the same nominal frequency of 1. We also assume that the communication at any link has a nominal speed of 1. In our system model we focus on the process variations on each individual core, and we assume there is no process variation on the links.

The *nominal design* of application G based on the logical architecture $A_{r \times c}^l$ (denoted as $\mathcal{N}(G, A^l)$) is defined by the mapping between the task nodes in G and processor cores in $A_{r \times c}^l$. That is

$$\mathcal{N}(G, A^l) = \{(v_i, C_{x,y}^l) | v_i \text{ is assigned to core } C_{x,y}^l\}, \quad (3.2)$$

$$i = 1, \dots, k; \quad (3.3)$$

$$0 \leq x \leq r - 1;$$

$$0 \leq y \leq c - 1.$$

We assume the traditional NoC mesh network architecture and the deterministic X-Y routing algorithm [61] is used.

We define the *physical architecture* (denoted as $A_{m \times n}^p$) as a $m \times n$ mesh architecture, i.e.

$$A_{m \times n}^p = \{C_{i,j}^p, i = 0, \dots, m - 1; j = 0, \dots, n - 1\}. \quad (3.4)$$

where $C_{i,j}^p$ represents the core located at position (i, j) in the physical mesh architecture. We use f_{ij} to represent the maximum clock frequency of core $C_{i,j}^p$, which is normalized to the nominal frequency of the logical core. $f_{ij} = 0$ indicates that core $C_{i,j}^p$ is a faulty core.

3.2.3 Problem Formulation

With the system model defined above, we are now ready to define our research problem.

Problem 3.2.1. *Given*

- *an application G ;*
- *a logical architecture $A_{r \times c}^l$;*
- *the nominal design of G on $A_{r \times c}^l$, i.e. $\mathcal{N}(G, A^l)$;*
- *the physical architecture $A_{m \times n}^p$ and its performance variations, i.e. $f_{ij}, i = 0, \dots, m - 1; j = 0, \dots, n - 1$,*

Find the mapping of $\mathcal{M} = \{C_{i,j}^l \rightarrow C_{x,y}^p | i = 0, \dots, r - 1; j = 0, \dots, c - 1; 0 \leq x \leq m - 1; 0 \leq y \leq n - 1\}$ such that the maximum latency to execute G based on $\mathcal{N}(G, A^l)$ is minimized.

3.3 Our Approach

In this section, we present three approaches to solve Problem 6.2.1 as defined above. The first approach is a simple heuristic that tries to match the logic node with the largest workload to the highest performance core in the physical architecture. The second and third approaches are developed based on the *opportunity cost*, a concept originated from the economics discipline. The second approach considers only the core performance. The third approach considers not only the core performance but also the communication cost.

3.3.1 A Simple Workload/Performance Matching Heuristic

Our goal is to map the logical topology to the physical topology such that an existing nominal design can achieve the best performance in the presence of faulty cores and performance variations on the physical topology. Since different cores may have different performances or processing speeds, an intuitive approach is therefore to match the logical core with the largest workload assignment to the physical core with the highest processing speed. The rationale behind this approach is that, the larger the workload is assigned to highest performance core, the more workload can be benefited from the highest processing speed. The algorithm, which we called *simple workload/performance matching* (SWPM) heuristic, is presented in Algorithm 1.

Algorithm 1 A simple heuristic to match high workload logical core to high performance physical core.

- 1: $\mathcal{M} = \emptyset$;
 - 2: $LC =$ The sorted list of $C_{i,j}^l \in A_{r \times c}^l, i = 0, \dots, r - 1; j = 0, \dots, c - 1$ by their workload based on nominal design $\mathcal{N}(G, A^l)$ in decreasing order;
 - 3: $LP =$ The sorted list of $C_{i,j}^p \in A_{m \times n}^p$ by $f_{ij}, i = 0, \dots, m - 1; j = 0, \dots, n - 1$ in decreasing order;
 - 4: **for** $i = 0$ to $sizeof(LC) - 1$ // for each logical core in the sorted list **do**
 - 5: **if** The total workload assigned to $LC(i) > 0$ **then**
 - 6: $\mathcal{M} = \mathcal{M} + \{LC(i) \rightarrow LP(i)\}$;
 - 7: **end if**
 - 8: **end for**
-

Algorithm 1 sorts the logic cores based on the assigned workloads and the physical cores based on their performances. Then, a logic core is mapped one by one to a physical core accordingly from the two lists. The complexity mainly comes from the sorting of the two lists. we assume the physical mesh is larger than the logical mesh. Therefore, the complexity of Algorithm 1 is $O((m \times n) \log(m \times n))$.

While Algorithm 1 is fast and intuitive, it has several issues. First, even though those high performance cores are used to speed up executions of larger workloads,

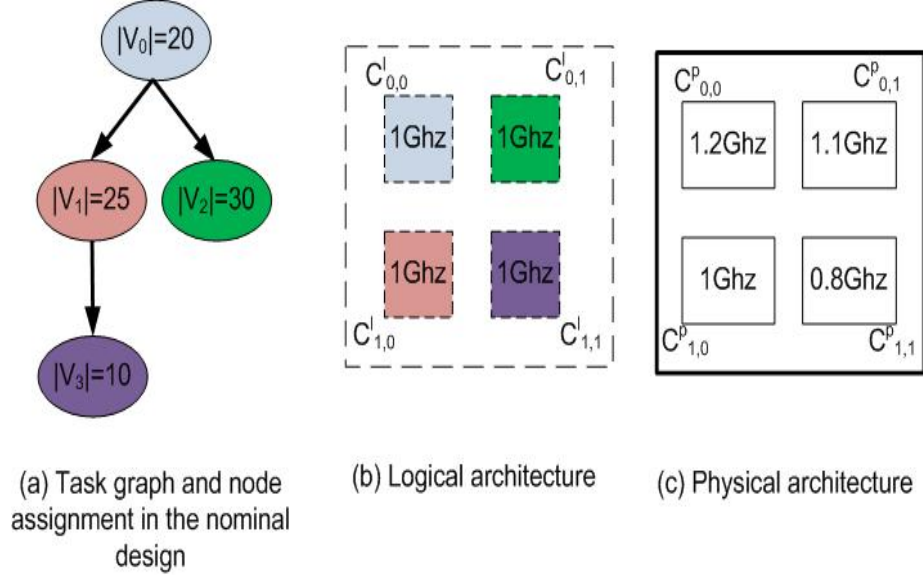


Figure 3.3: An illustrative example for opportunity cost and performance metric. The colors and shades of task nodes in (a) imply the corresponding assignment to the logical topology: $v_0 \rightarrow C_{0,0}^l$, $v_1 \rightarrow C_{1,0}^l$, $v_2 \rightarrow C_{0,1}^l$, $v_3 \rightarrow C_{1,1}^l$.

these workloads are not necessarily located on the *critical path*, i.e. the longest execution path of a task graph. In that case, the latency improvement when executing the task graph is limited. Second, Algorithm 1 considers only performance differences of different cores and do not take their locations into consideration. When two neighboring logical cores are separated far away in the physical mesh, the increased communication overhead can offset the performance improvement or even degrade the overall performance, or the latency when executing tasks. In what follows, we develop two new approaches to address these problems.

3.3.2 Opportunity Cost Based Workload/Performance Mapping

It is desirable to optimize the latency of the critical path to improve the performance when realizing the logical topology to the physical topology. In the meantime, how-

ever, optimizing the critical path too aggressively may cause other execution paths to become critical and thus degrade the optimization performance. The problem is then how to develop effective algorithms for logical/physical topology mapping that can optimize the maximum latency when executing a task graph. In what follows, we discuss a heuristic developed for this goal. For the sake of simplicity, we first assume the communication cost is negligible.

When designing a highly effective logical to physical topology mapping, one critical problem is how to evaluate the impact or performance of a decision when mapping a logical node to a physical node. Note that it is not difficult to prove that Problem 6.2.1 is in fact NP-hard. In our approach, we resort to adopting the *opportunity cost* as the metric to make our decisions. The opportunity cost is the cost of any activity measured in terms of the value of the next best alternative forgone (that is not chosen). It is the sacrifice related to the second best choice available to someone, or group, who has picked among several mutually exclusive choices. For more details about the opportunity cost, readers can refer to [26] or other related references.

We use a simple illustrative example to explain the concept of the opportunity cost and its use in designing our performance metric. Figure 3.3(a) shows a task graph with four nodes. The colors and shades represent their assignments to the logical topology as shown in Figure 3.3(b). To calculate the latency when executing a task graph, we assume that, if a logical core has not been mapped, all the task nodes assigned to this logical core take their nominal execution time; if a logical core has already been mapped, then new execution times on the practical cores will be used.

Now consider the decision of mapping logical core $C_{0,0}^l$ to physical core $C_{0,0}^p$. The task graph latency of this mapping is 51.67. Since the latency in the nominal design

Algorithm 2 Workload/performance mapping based on opportunity cost.

- 1: $\mathcal{M} = \emptyset$;
 - 2: $LC =$ The list of $C_{i,j}^l \in A_{r \times c}^l, i = 0, \dots, r - 1; j = 0, \dots, c - 1$ with workload assignments larger than 0;
 - 3: $LP =$ The list of $C_{i,j}^p \in A_{m \times n}^p$, excluding faulty cores;
 - 4: **while** $LC \neq \emptyset$ **do**
 - 5: Find $C_{i,j}^l \in LC$ and $C_{x,y}^p \in LP$ such that $\mathcal{P}(C_{i,j}^l \rightarrow C_{x,y}^p)$ is maximized;
 - 6: $\mathcal{M} = \mathcal{M} + \{C_{i,j}^l \rightarrow C_{x,y}^p\}$;
 - 7: Remove $C_{i,j}^l$ and $C_{x,y}^p$;
 - 8: **end while**
-

is 55, we define that the *profit* of this decision is $55 - 51.67 = 3.33$. For the rest of the alternatives to map logical core $C_{0,0}^l$, the best choice is to map it to $C_{0,1}^p$ with latency of 53.18. The corresponding profit is $55 - 53.18 = 1.82$, which is the opportunity cost to map $C_{0,0}^l$ to $C_{0,0}^p$. We thus define the performance of the decision as the difference of its profit and opportunity cost, or $3.33 - 1.82 = 1.51$. In what follows, we formally define the performance metric used in our approach.

Definition 3.3.1. *Given a decision to map logical core $C_{i,j}^l$ to physical core $C_{x,y}^p$, i.e. $C_{i,j}^l \rightarrow C_{x,y}^p$, let its profit be denoted as $Prof(C_{i,j}^l \rightarrow C_{x,y}^p)$, and let its opportunity cost (i.e. the performance associated with the best choice to map $C_{i,j}^l$ other than $C_{x,y}^p$) be denoted as $OC(C_{i,j}^l \rightarrow C_{x,y}^p)$. Then the performance of the decision, denoted as $\mathcal{P}(C_{i,j}^l \rightarrow C_{x,y}^p)$, is defined as*

$$\mathcal{P}(C_{i,j}^l \rightarrow C_{x,y}^p) = Prof(C_{i,j}^l \rightarrow C_{x,y}^p) - OC(C_{i,j}^l \rightarrow C_{x,y}^p). \quad (3.5)$$

Specifically, for the example in Figure 3.3, we have $\mathcal{P}(C_{0,0}^l \rightarrow C_{0,0}^p) = 1.51$, $\mathcal{P}(C_{0,1}^l \rightarrow C_{0,0}^p) = 0$, $\mathcal{P}(C_{1,0}^l \rightarrow C_{0,0}^p) = 1.9$, and $\mathcal{P}(C_{1,1}^l \rightarrow C_{0,0}^p) = 0.76$. It is interesting to note that, according to Definition 3.3.1, mapping the logical core with the largest workload assignment (i.e. $C_{0,1}^l$) to the fastest core (i.e. $C_{0,0}^p$) does not reduce the critical path latency and thus has the lowest performance.

After establishing the metric to evaluate a mapping decision, we are ready to introduce our heuristic algorithm, which is given in Algorithm 2. The most critical section of Algorithm 2 is the while loop, which selects the mapping with the largest performance according to equation (3.5). In the worst case, the complexity of the while loop is $O(kmn)$ since $m \times n$ different mappings need to be checked, and the complexity to obtain the latency for a task graph is k , where k is the number of task nodes. In the worst case, the while loop will be executed for $r \times c$ times. Therefore, the overall complexity of Algorithm 2 is $O(krcmn)$.

3.3.3 Logical/Physical Topology Mapping With Communication Awareness

Neither Algorithm 1 nor Algorithm 2 takes the communication cost into consideration. They work fine when the communication cost is really small and negligible. When the communication cost becomes significant, especially for many-core platforms, the qualities of the mapping results by Algorithm 1 and Algorithm 2 can be severely compromised. In what follows, we propose an iterative algorithm (shown in Algorithm 3) to improve the performance of existing mapping results while taking the communication into consideration.

In principle, Algorithm 3 uses similar performance metric based on opportunity cost to evaluate a mapping decision. When calculating the latency for the task graph, the communication cost based on XY-routing can be incorporated into the calculation of performance of a mapping decision. Another major difference between Algorithm 3 and Algorithm 2 is that Algorithm 3 can iteratively improve the mapping solution, until the improvement threshold defined by the user can be satisfied.

Algorithm 3 Logical/Physical mapping with communication cost awareness.

```

1: Initialize  $\mathcal{M}_0$ ; // by Algorithm 1 or 2
2:  $L_{orig}$  = latency of executing  $G$  based on  $\mathcal{M}_0$ ;
3: Improvement = 0;
4: while Improvement  $< \epsilon$  // user defined threshold do
5:    $LC$  = The list of  $C_{i,j}^l \in A_{r \times c}^l, i = 0, \dots, r - 1; j = 0, \dots, c - 1$  with work
      assignment larger than 0;
6:    $LP$  = The list of  $C_{i,j}^p \in A_{m \times n}^p$ , excluding faulty cores;
7:    $\mathcal{M} = \emptyset$ ;
8:   while  $LC \neq \emptyset$  do
9:     Find  $C_{i,j}^l \in LC$  and  $C_{x,y}^p \in LP$  such that  $\mathcal{P}(C_{i,j}^l \rightarrow C_{x,y}^p)$  is maximized;
10:     $\mathcal{M} = \mathcal{M} + \{C_{i,j}^l \rightarrow C_{x,y}^p\}$ ;
11:    Remove  $C_{i,j}^l$  and  $C_{x,y}^p$ ;
12:  end while
13:   $L_{new}$  = latency of executing  $G$  based on  $\mathcal{M}_0$ ;
14:  Improvement =  $\frac{L_{orig} - L_{new}}{L_{orig}}$ ;
15:   $L_{orig} = L_{new}$ ;
16: end while

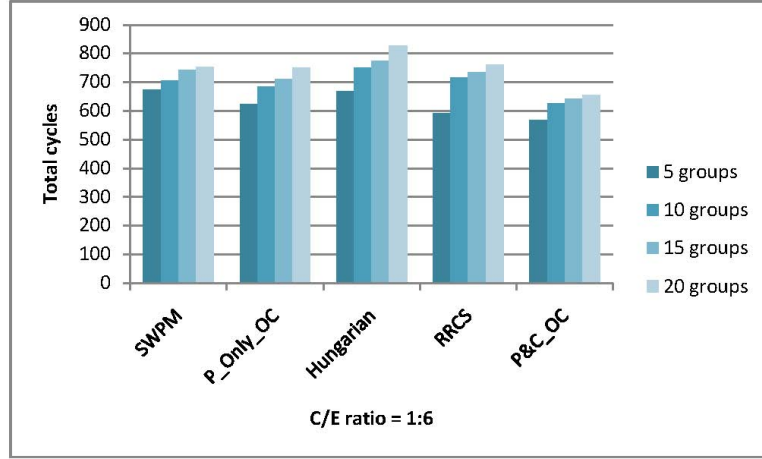
```

The complexity of the while loop from line 8 to 12 is similar to that in Algorithm 2.

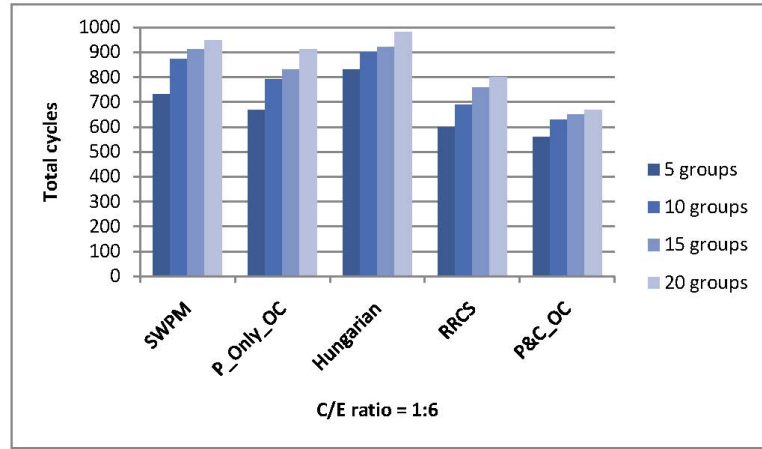
The overall complexity of Algorithm 3 depends on the exact value of ϵ .

3.4 Experimental Results

In this section, we perform three experiments to study the performance of three approaches we presented in the previous section. For ease of presentation, we use *SWPM* to denote Algorithm 1, *P_Only_OC* for Algorithm 2, and *P&C_OC* for Algorithm 3. We also compare our algorithms with two previous works, i.e. the *RRCS* algorithm [178] and the *Hungarian* algorithm [175]. The *RRCS* algorithm intends to replace the faulty cores and reshape the mesh to mirror the logical mesh while the *Hungarian* algorithm tries to re-map the physical mesh to logical mesh to minimize the communication changes. We investigated the performance of these



(a) 5×6 mesh



(b) 10×11 mesh

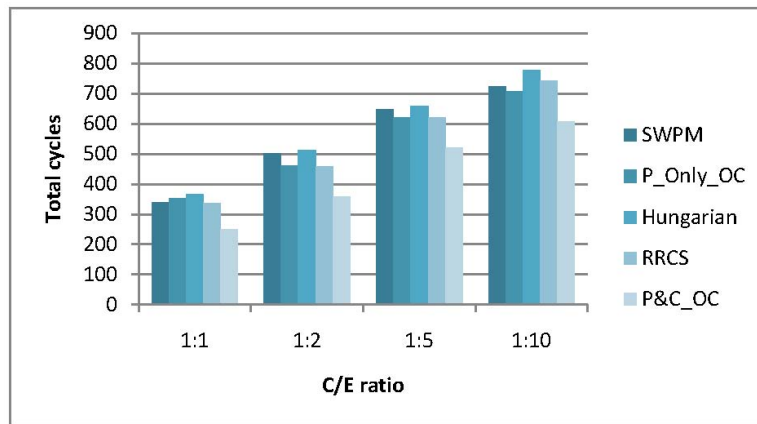
Figure 3.4: Performance vs. different mesh sizes and different group numbers

five different approaches under different mesh sizes, numbers of task nodes, communication/execution ratios, as well as their computational costs.

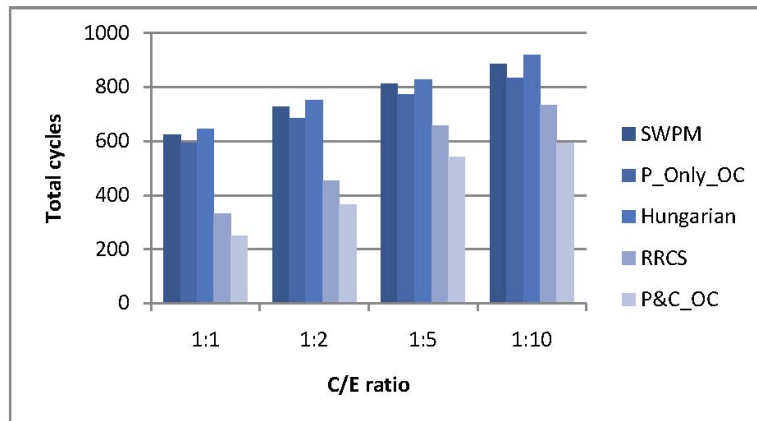
3.4.1 Experimental Setup

In our simulation study, we used TGFF [41] to randomly generate task graphs (60 nodes) and also randomly cluster task nodes into groups and map to different logical cores, from a 5×6 and a 10×11 mesh. The reason we used $n \times (n + 1)$ mesh

is because the *RRCS* and the *Hungarian* algorithms assume such topology. The communication of each edge and execution time of each task are randomly generated. The frequency of each processor is also randomly generated using normal distribution with mean value $\mu = 1$, i.e. the nominal frequency, and variance value $\sigma = 0.1$, based on [24]. Unless specified otherwise, we assume the *P&C_OC* algorithm stops after 200 iterations. All experiments were running on a Window XP/SP3 platform powered by Intel(R) Core(TM)2 Duo CPU @ 2.93GHz with 3.21 GB of RAM.



(a) 5×6 mesh



(b) 10×11 mesh

Figure 3.5: Performance vs. different communication/execution ratios

3.4.2 Performance vs. Mesh Sizes And Group Numbers

In this experiment, we compared the performance of different algorithms under different size meshes: 5×6 and 10×11 . The execution time of a task node was randomly generated from interval $[10:50]$. The communication cost of an edge was randomly chosen from interval $[1:10]$. The average results among all test cases were collected and plotted in Figure 3.4.

From Figure 3.4, we can see that *P_Only_OC* consistently outperformed *SWPM* under different mesh sizes and different group numbers. For example, for mesh size of 5×6 and task group number of 5, we can see that *P_Only_OC* outperformed *SWPM* as much as 10% in latency reduction. This is because *SWPM* optimizes aggressively on the logical cores with large workload assignments. Unfortunately, as indicated in our previous illustrative example (Figure 3.3), the overall performance can be severely limited if the workloads are not located on the critical path. *P_Only_OC*, on the other hand, judiciously chooses logical/physical mapping based on application characteristics and thus can outperform *SWPM*.

When comparing *P_Only_OC* and *RRCS*, it is interesting to see that *P_Only_OC* performs better than *RRCS* for small meshes but becomes worse than *RRCS* when the mesh size is large or the task group number is small. For example, for mesh size of 5×6 and group size of 10, *P_Only_OC* outperformed *RRCS* by approximately 4%. For large mesh size of 10×11 , *RRCS* can outperform *P_Only_OC* by as much as 12.5%. This is because *P_Only_OC* can take the application information into consideration and outperform *RRCS*. However, our experimental results also indicate that *P_Only_OC* works only in small mesh size. For large mesh sizes, the *P_Only_OC* algorithm can potentially distribute tasks far away from each other and therefore degrade the overall performance. And the *Hungarian* algorithm is the worst one as we have discussed previously; it is good for small mesh size, i.e.

5×5 and small number of faulty cores, i.e. no more than 4 faulty cores. However, in our setup, we assume 5 and 10 faulty cores for 5×6 mesh and 10×11 mesh, respectively. The *Hungarian* algorithm always tries to re-map the faulty cores using the redundant cores which are aligned to the rightmost column of the mesh, therefore, it results in poor performance.

Finally, we can see that *P&C_OC* consistently outperforms other algorithms for different mesh sizes and groups, and the results improved with the growth of mesh size and number of task groups. From Figure 3.4, on average *P&C_OC* can outperform *RRCS* by 13% and 16% for mesh size of 5×6 and 10×11 , respectively. The experimental results greatly highlight the excellent performance of *P&C_OC*.

3.4.3 Performance vs. Different Communication/Execution Ratios

Next, we study how communication cost can affect the performance of different approaches. Let communication cost be generated within interval $[a,b]$ and execution time of task node be generated within interval $[c,d]$, the C/E ratio can be defined in Equation 3.6.

$$ratio = \frac{b + a}{d + c}, \quad (3.6)$$

We randomly generated different test cases with different C/E ratio and tested the four algorithms mentioned above. The C/E ratios were set to be 1:1, 1:2, 1:5, 1:10. The results for different test cases were collected and plotted in Figure 3.5.

From Figure 3.5(b), the improvement of *P&C_OC* over *P_Only_OC* increases as communication cost increases. When communication cost is much less than the execution cost (C/E ratio = 1:10), *P&C_OC* improves upon *P_Only_OC* about

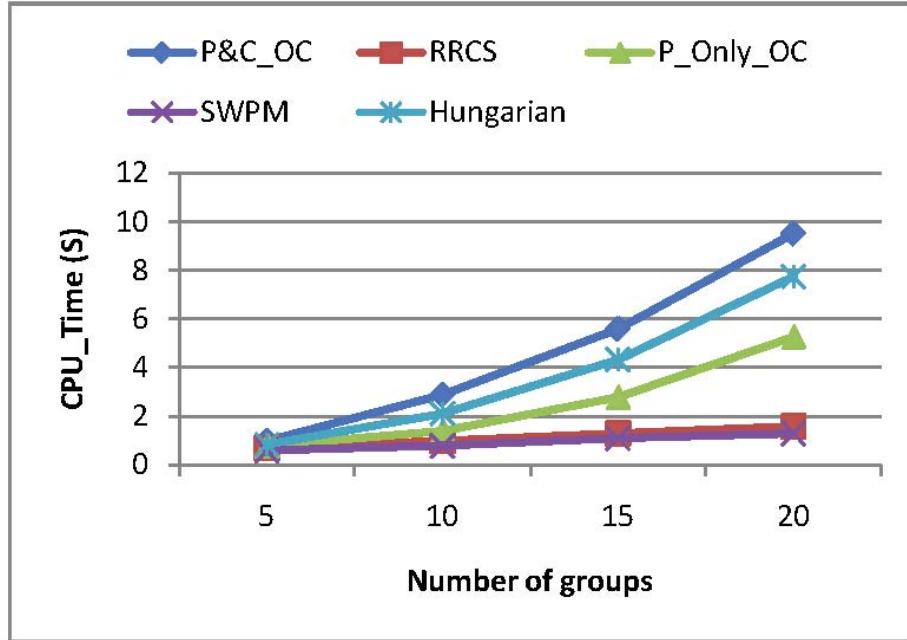


Figure 3.6: Computational Time comparisons for different algorithms on 10×11 mesh

30%. When the communication cost is almost comparable to the execution cost (C/E ratio = 1:1), the average latency by *P_Only_OC* is more than double that by *P&C_OC* and *SWPM*, i.e. approaches that do not take the communication into consideration.

3.4.4 Computational Cost

We next studied the computational cost for each algorithm on mesh size of 10×11 . Figure 3.6 shows the computation times with different numbers of task groups for the five algorithms. For the *P&C_OC* algorithm, we set the threshold to 16%. It is not surprising to see that *SWPM* and *RRCS* have computational costs nearly linear to the task group, while *P_Only_OC*, *Hungarian* and *P&C_OC* are increasing very fast, as discussed before.

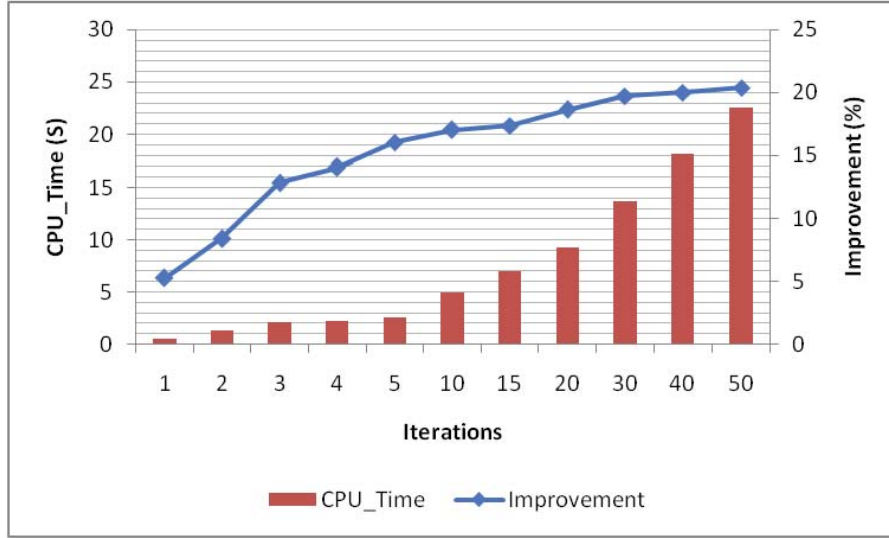


Figure 3.7: Computational Time and Improvement comparisons for different iterations on 10×11 mesh

To further understand the computational complexity of *P&C_OC*, we conducted another set of experiments with 20 task groups and kept track of the solution quality for each iteration. As shown in Figure 3.7, we can see that from the first iteration all the way to the 50th iteration, the CPU time increases rapidly. The improvement also grows rapidly during the first several iterations until it reaches around 22% in improvement and becomes saturated. How to speedup the *P&C_OC* without compromising its solution quality is an interesting problem worthy of future study.

3.5 Summary

Performance variations can reduce the fabrication yield and degrade the quality of the nominal design. Different from previous research at the device level, during the post-fabrication, or the statistical approach at the system level, we propose to deal with the process variations when deploying the nominal design to a dedicated device. We introduce a framework to judiciously reconfigure the underlying physical

architecture to mirror the logical architecture and maximize the performance of the nominal design. Heuristics based on the concept of opportunity cost are introduced in this work. From our experimental studies, the proposed approach can achieve up to 30% and with an average 15% of performance improvement (i.e. schedule length) by taking advantage of the heterogeneity of each individual platform.

CHAPTER 4

HETEROGENEITY EXPLORATION FOR PEAK TEMPERATURE REDUCTION ON MULTI-CORE PLATFORMS

In the previous chapter, we addressed process variation-aware scheduling to minimize the scheduling length of a task graph. In this chapter, we extended our previous work in Chapter 3 to optimize the peak temperature of a real-time schedule on multi-core platforms under process variation. We develop three computationally efficient algorithms for deploying applications to individual devices. Our simulation study has clearly shown that, by taking advantage of the uniqueness of each individual physical chip, the proposed approaches significantly reduce the peak temperature. The experiments also show that these approaches are efficient and have low operational cost.

4.1 Related Work

Many approaches have been proposed to deal with process variation problems. Significant achievements have been made on layout techniques and other device technologies by adding built-in sensors or redundant devices [94, 129, 95]. However, it becomes increasingly challenging as transistor size scales towards dimensions close to or below 10 nm [154]. Besides extensive work on layout and device level, there are increasing research efforts to address the process variation problem from architecture and system level. For example, performance binning technique is proposed to cluster chips with similar performance [141, 122]. However, this approach cannot deal with process variation of different cores within the same chip. Another popular approach is to adopt the statistical approach in the design optimization process. As an example, Wang et al. [163] proposed a task mapping and scheduling algorithm to maximize the performance yield rate (i.e., the probability that a processor can

meet the desired performance of a given application) by statistically taking both variations of cores and physical links into account. These statistical approaches try to optimize results in a probabilistic manner. When deploying the design to each individual processor, the designs can be either too optimistic or too pessimistic due to different performance variations. One recent work [135] exploited process variations in Dark-Silicon homogeneous chip multi-processors, however they only considered the frequency variations and ignored leakage variations between cores.

There is another interesting approach proposed to address the process variation problem. This approach, so called *topology virtualization* [167], calls for judiciously mirroring the physical architecture of an individual device to the logic architecture of an application when the application is deployed (installed/initiated) to the device. Figure 4.1 illustrates this approach.

Assume an application is developed based on a 3×3 logical homogeneous multi-core mesh architecture. The physical chip, on the other hand, is not necessarily the same 3×3 architecture. To fight for the process variation problem, it has been a common practice in industry to add redundant resources (e.g. processing cores) so that the entire chip can still work even if some cores are faulty [129, 94]. Assume the physical architecture of a chip is a mesh of 3×4 as shown in Figure 4.1. Note that, due to the process variation problem, the performance of all these cores is not necessarily homogeneous. Moreover, the design based on the logical architecture does not necessarily utilize each processor in exactly the same way. Opportunity is thus presented to optimize the performance of original nominal design by mirroring the physical architecture of each individual chip to the logical architecture differently as shown in Figure 4.1. There are a number of distinct advantages to this approach. First, compared with the statistical approach, this approach can exploit the unique characteristic of each individual device and better optimize the system performance.

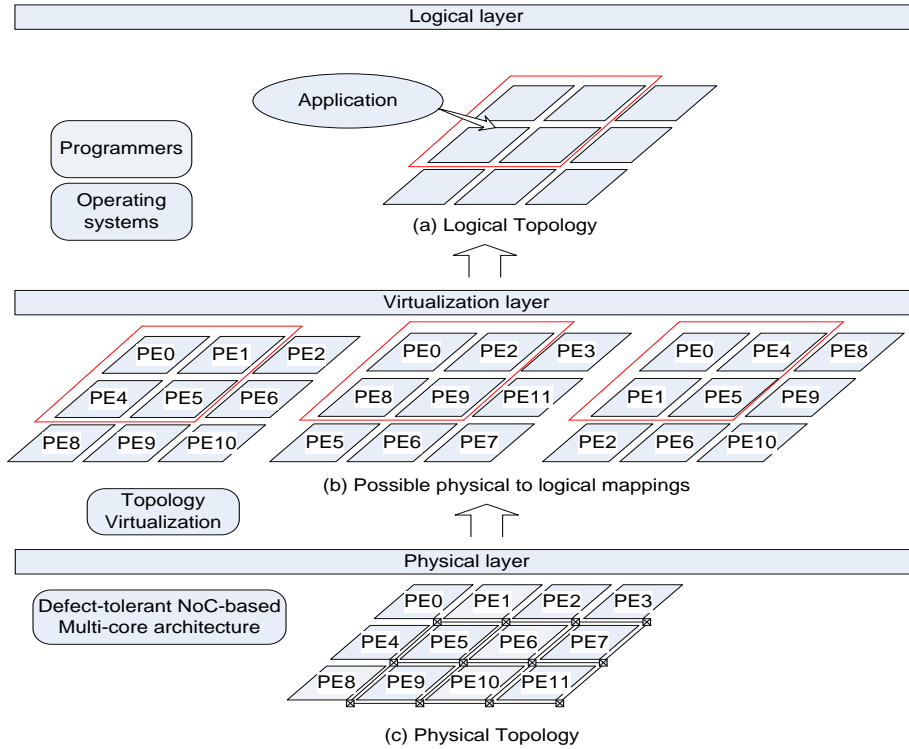


Figure 4.1: The topology virtualization framework. (a) Target application designed based on the nominal parameters of a 3×3 logical topology; (b) Configure the practical topology of an individual processor differently to mirror the logical topology and optimize the performance of the target application; (3) The physical topology, a 3×4 mesh, for the processor.

Second, the architecture changes can be managed by the operating system or lower level software such as BIOS, which is totally transparent to the application software.

We believe that heterogeneity due to process variation, if explored properly, can in fact improve the design objective of a real-time application. In this work, we study the problem of how to reduce the peak temperature by exploiting the architecture heterogeneity due to process variation. A few works are closely related to our approach proposed in this chapter. When a processor has faulty cores and more redundant cores, Zhang et al. [176, 178] proposed several heuristics to replace faulty cores with redundant cores to improve the fabrication yield. They further extended

their work to deal with performance variations by constructing sub-meshes using cores with similar performance [179]. Yue et al. [174, 175] improved upon Zhang’s work [176, 178] by taking application characteristics into consideration, and intended to maintain the similar real-time performance after replacing faulty cores with redundant cores. A more recent work by Wang et al. [167] considered process variation on homogeneous multi-core platforms and proposed three re-mapping heuristics to maximize the throughput of task graph.

It is not difficult to see that the problem to optimize the performance of an application by mirroring the physical topology to the logical topology is an NP-hard problem [53]. While it is a common practice to use certain time-consuming techniques such as Genetic Algorithm (GA) [105, 138] and/or Simulated Annealing Algorithm (SAA) [176, 178] to solve this problem, this is not viable for the *topology virtualization* approach. Since the physical to logical topology mapping is performed when deploying (installing or initiating) the application software on an individual device, the key to the success of this approach is to develop computationally efficient mapping methods that can effectively optimize the performance metrics for application software. To this end, we developed three physical to logical topology mapping heuristics to reduce the peak temperature of a processor. Our simulation study shows that the *topology virtualization* approach is very effective in reducing the peak temperature for a processor.

In what follows, we introduce the system, power and thermal models the research is based upon, and formulate the problem we are to address in Section 4.2. In Section 4.3, We discuss how to rapidly calculate the peak temperature for a periodic application . We then present three computationally efficient heuristics to minimize peak temperature in Section 4.4. Experimental results are discussed in Section 4.5, and we conclude in Section 4.6.

Nominal Value (V, F)	Core 0	Core 1
(0.8, 0.8)	(0.8, 0.76)	(0.8, 0.82)
(0.9, 0.9)	(0.9, 0.87)	(0.9, 0.93)
(1.0, 1.0)	(1.0, 0.95)	(1.0, 1.04)
(1.1, 1.1)	(1.1, 1.03)	(1.1, 1.13)
(1.2, 1.2)	(1.2, 1.18)	(1.2, 1.25)

Figure 4.2: Voltage-Frequency variation example between cores.

4.2 Preliminary

In this section, we first introduce the system model, power and thermal models and then formulate the research problem we are to address in this work.

4.2.1 System Models

The multi-core platform considered in this work consists of identical cores with traditional 2-D mesh architecture. Each core can run in one of r different operating modes. Each running mode is characterized by a tuple (v_k, f_k) ($1 \leq k \leq r$), where v_k is the supply voltage and f_k is the working frequency for mode k , respectively. Note that due to manufacture variations, cores may have different maximum frequencies deviated from their nominal value. Therefore, the frequency can also behave differently in cores that are running with the same supply voltage. Figure 4.2 shows an example of two cores with their voltage-frequency modes compared to the nominal values (parameters are generated based on [94]).

Specifically, we assume the *physical architecture* (denoted as $A_{m \times n}^p$) is an $m \times n$ mesh, i.e.

$$A_{m \times n}^p = \{\mathcal{A}_{i,j}^p, i = 0, \dots, m - 1; j = 0, \dots, n - 1\}. \quad (4.1)$$

where $\mathcal{A}_{i,j}^p$ represents the core located at position (i, j) in the physical topology. The target application is periodic with period of L . We assume the original *nominal*

design of the application is based on a *logical architecture* (denoted as $A_{x \times y}^l$), which forms a standard $x \times y$ homogeneous mesh architecture, i.e.

$$A_{x \times y}^l = \{\mathcal{A}_{i,j}^l, i = 0, \dots, x - 1; j = 0, \dots, y - 1\}. \quad (4.2)$$

where $\mathcal{A}_{i,j}^l$ represents the core located at position (i, j) in the logical topology. The *nominal design* $\mathbf{S}(t)$ consists of a set of static, periodic voltage schedules, each of which, i.e. $\mathbf{S}_i(t)$, is applied to one logical core and dictates the change of its processing speed in each period. We assume that each schedule consists of a set of non-overlapping intervals with total length of L . Each interval has its own specified running modes. Let the voltage schedule on core i , denoted as $\mathbf{S}_i(t)$, consist of a set of intervals $[t_0, t_1], \dots, [t_{q-1}, t_q]$ such that $\bigcup_{q=1}^s [t_{q-1}, t_q] = [0, L]$, and $[t_{q-1}, t_q] \cap [t_{p-1}, t_p] = \emptyset$, if $q \neq p$. Also, let the running modes of interval i be $[v_i, f_i]$. We define the *utilization* of core i , denoted as U_i as

$$U_i = \frac{\sum_i (t_i - t_{i-1}) \times v_i}{v_{max} \times L}. \quad (4.3)$$

4.2.2 Power And Thermal Models

The total power dissipation of each core contains two parts: dynamic power and leakage power. We assume that the dynamic power is independent of temperature but sensitive to variation while the leakage power is sensitive to both. The total power dissipation of core i , denoted as P_i , is formulated as:

$$P_i = P_i^{dym} + P_i^{leak}, \quad (4.4)$$

$$P_i^{dym} = \gamma_{k_i} \cdot v_{k_i}^2 \cdot f_{k_i}, \quad (4.5)$$

$$P_i^{leak} = (\alpha_{k_i} + \beta_{k_i} \cdot T_i(t)) \cdot (v_{k_i} + \Delta_{leak}^i). \quad (4.6)$$

where α_{k_i} , β_{k_i} and γ_{k_i} are constants that depend on the mode k_i of core i . Δ_{leak}^i is a given constant that models the leakage power variations due to the impact of die-to-die (D2D) and within-die (WID) process variation [58].

We use the RC network to model the thermal behavior of a multi-core platform, same to that in [145, 160]. Let C_i and R_{ij} denote the thermal capacitance (in $Watt/^\circ C$) of core i and thermal resistance (in $J/^\circ C$) between core i and j , respectively. The thermal behavior of the i^{th} core can be formulated as

$$C_i \cdot \frac{dT_i(t)}{dt} + \frac{T_i(t)}{R_{ii}} + \sum_{j \neq i} \frac{T_i(t) - T_j(t)}{R_{ij}} = P_i(t) \quad (4.7)$$

Incorporating equation (5.11) in the above equation, we have

$$C_i \cdot \frac{dT_i(t)}{dt} + G_{ii} \cdot T_i(t) + \sum_{j \neq i} G_{ij} \cdot T_j(t) = \Psi_i \quad (4.8)$$

where

$$G_{ij} = \begin{cases} \sum_{j=1}^m \frac{1}{R_{ij}} - \beta_{k_i} \cdot v_{k_i} - \beta_{k_i} \Delta_{leak}^i & \text{if } i = j \\ \frac{-1}{R_{ij}} & \text{otherwise} \end{cases} \quad (4.9)$$

and

$$\Psi_i = \alpha_{k_i} \cdot v_{k_i} + \alpha_{k_i} \Delta_{leak}^i + \gamma_{k_i} \cdot v_{k_i}^2 \cdot f_{k_i} \quad (4.10)$$

Let \mathbf{T}_{amb} denote the ambient temperature. We thus have

$$\mathbf{C} \frac{d\mathbf{T}(t)}{dt} + \mathbf{G}(\mathbf{T}(t) - \mathbf{T}_{amb}) = \mathbf{\Psi} \quad (4.11)$$

where \mathbf{C} and \mathbf{G} are $m \times m$ matrices while $\mathbf{T}(t)$ and $\mathbf{\Psi}$ are $m \times 1$ vectors.

$$\mathbf{C} = \begin{bmatrix} C_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & C_m \end{bmatrix} \quad \mathbf{G} = \begin{bmatrix} G_{11} & \cdots & G_{1m} \\ \vdots & \ddots & \vdots \\ G_{m1} & \cdots & G_{mm} \end{bmatrix} \quad (4.12)$$

$$\mathbf{T}(t) = \begin{bmatrix} T_1(t) \\ \vdots \\ T_m(t) \end{bmatrix} \quad \mathbf{\Psi} = \begin{bmatrix} \Psi_1 \\ \vdots \\ \Psi_m \end{bmatrix} \quad (4.13)$$

4.2.3 Problem Formulation

With all models discussed above, our problem is to minimize peak temperature while guaranteeing timing constraints. It is worth pointing out that the nominal design $\mathbf{S}(\mathbf{t})$ is based on manufacture-variation-free scenario, hence, when we apply the nominal design to each individual chip which may be affected by manufacture variations, we may not be able to guarantee timing constraints if no appropriate actions are taken. We formally define the research problem below.

Problem 4.2.1. *Given*

- *a physical topology of a multi-core platform $A_{m \times n}^p$, r different processor modes and leakage variation Δ_{leak} for each core;*
- *a logical topology $A_{x \times y}^l$;*
- *a nominal design $\mathbf{S}(\mathbf{t})$,*

determine the physical to logical topology mapping such that the chip peak temperature of the chip is minimized when running $\mathbf{S}(\mathbf{t})$ on $A_{m \times n}^p$ and all timing constraints are also met.

4.3 Temperature Dynamics Formulation

Our goal is to minimize the peak temperature when running the *nominal design* on the practical processor. To this end, it is necessary that we can quickly identify the exact peak temperature when running a periodic schedule. In what follows, we introduce a method to quickly calculate the peak temperature for a periodic voltage schedule on multi-core platforms.

Consider an interval $[t_{q-1}, t_q]$ and assume the supply voltages or working frequencies of all cores remain the same within the interval. Let κ_q represent the specific

running modes of all cores in interval $[t_{q-1}, t_q]$. Then based on equation (4.11), we have

$$\left. \frac{d\mathbf{T}(t)}{dt} \right|_{t \in [t_{q-1}, t_q]} = \mathbf{A}_{\kappa_q} (\mathbf{T}(t) - T_{amb}) + \mathbf{B}_{\kappa_q} \quad (4.14)$$

where $\mathbf{A}_{\kappa_q} = -\mathbf{C}^{-1} \mathbf{G}_{\kappa_q}$ and $\mathbf{B}_{\kappa_q} = \mathbf{C}^{-1} \mathbf{\Psi}_{\kappa_q}$. Since \mathbf{A}_{κ_q} and \mathbf{B}_{κ_q} are constant, equation (4.14) is simply a first-order constant coefficient ordinary differential equation (ODE) with the following solution:

$$\mathbf{T}(t_q) = e^{\mathbf{A}_{\kappa_q} \Delta t_q} (\mathbf{T}(t_{q-1}) - T_{amb}) + \mathbf{A}_{\kappa_q}^{-1} (e^{\mathbf{A}_{\kappa_q} \Delta t_q} - I) \mathbf{B}_{\kappa_q} + T_{amb} \quad (4.15)$$

where $\Delta t_q = t_q - t_{q-1}$. Therefore, given a state interval, its ending temperature can be determined by the starting temperature $\mathbf{T}(t_{q-1})$ and the corresponding interval mode κ_q .

With equation (4.15), given a periodic schedule $\mathbf{S}(t)$ and the initial temperature $\mathbf{T}(0)$, we can calculate the temperature at any time instant by tracing temperature from one interval to another. However, it can be computationally costly to trace the temperature until it reaches the steady state. It is also desirable to calculate the stable temperature by setting $\frac{d\mathbf{T}(t)}{dt} = 0$. This works if all cores run at a constant speed schedule, but does not work anymore for a periodic schedule with different running modes. In what follows, we present a fast method to identify the peak temperature for a periodic schedule $\mathbf{S}(t)$.

Let us first consider the temperature variation at the end of each period, i.e. $t = nL$. Let the scheduling points of $\mathbf{S}(t)$ in the first period be t_0, t_1, \dots, t_{s-1} , respectively. We assume that the running modes for all cores remain unchanged between two neighboring scheduling points. Similarly, let the *corresponding* scheduling points in the second period be $t'_0, t'_1, \dots, t'_{s-1}$, respectively. Note that $t_0 = 0, t'_0 = t_s = L$ and $t'_s = 2L$. According to equation (4.15), at time t_1 and t'_1 , we have

$$\mathbf{T}(t_1) = e^{\mathbf{A}_{\kappa_1} \Delta t_1} (\mathbf{T}(t_0) - T_{amb}) + \mathbf{A}_{\kappa_1}^{-1} (e^{\mathbf{A}_{\kappa_1} \Delta t_1} - I) \mathbf{B}_{\kappa_1} + T_{amb} \quad (4.16)$$

$$\mathbf{T}(t'_1) = e^{\mathbf{A}_{\kappa_1} \Delta t'_1} (\mathbf{T}(t'_0) - T_{amb}) + \mathbf{A}_{\kappa_1}^{-1} (e^{\mathbf{A}_{\kappa_1} \Delta t'_1} - \mathbf{I}) \mathbf{B}_{\kappa_1} + T_{amb} \quad (4.17)$$

Subtract equation (4.16) from (4.17), and simplify the result by applying $\Delta t'_1 = \Delta t_1$, $t_0 = 0$ and $t'_0 = L$, we get

$$\mathbf{T}(t'_1) - \mathbf{T}(t_1) = e^{\mathbf{A}_{\kappa_1} \Delta t_1} (\mathbf{T}(L) - \mathbf{T}(0))$$

Similarly, we can derive that

$$\begin{aligned} \mathbf{T}(t'_2) - \mathbf{T}(t_2) &= e^{\mathbf{A}_{\kappa_2} \Delta t_2} e^{\mathbf{A}_{\kappa_1} \Delta t_1} (\mathbf{T}(L) - \mathbf{T}(0)) \\ &\dots \\ \mathbf{T}(t'_s) - \mathbf{T}(t_s) &= e^{\mathbf{A}_{\kappa_s} \Delta t_s} \dots e^{\mathbf{A}_{\kappa_1} \Delta t_1} (\mathbf{T}(L) - \mathbf{T}(0)) \end{aligned} \quad (4.18)$$

Since $t_s = L$, $t'_s = 2L$, and let $\mathbf{K} = e^{\mathbf{A}_{\kappa_s} \Delta t_s} \dots e^{\mathbf{A}_{\kappa_1} \Delta t_1}$, equation (4.18) can be rewritten as

$$\mathbf{T}(2L) - \mathbf{T}(L) = \mathbf{K}(\mathbf{T}(L) - \mathbf{T}(0)) \quad (4.19)$$

Similarly, we have

$$\mathbf{T}(xL) - \mathbf{T}((x-1)L) = \mathbf{K}^{x-1}(\mathbf{T}(L) - \mathbf{T}(0)) \quad (4.20)$$

where $x = 1, 2, \dots, n$. Sum up these n equations, we get

$$\mathbf{T}(nL) = \mathbf{T}(0) + \left(\sum_{x=1}^n \mathbf{K}^{x-1} \right) (\mathbf{T}(L) - \mathbf{T}(0)) \quad (4.21)$$

In the above, $\{\mathbf{K}^{x-1} | x = 1, 2, \dots, n\}$ forms a matrix geometric series. If $(\mathbf{I} - \mathbf{K})$ is invertible, then we have

$$\mathbf{T}(nL) = \mathbf{T}(0) + (\mathbf{I} - \mathbf{K})^{-1} (\mathbf{I} - \mathbf{K}^n) (\mathbf{T}(L) - \mathbf{T}(0)) \quad (4.22)$$

Similarly, for any time instant $t = nL + t_q$, we can get that

$$\mathbf{T}(nL + t_q) = \mathbf{T}(t_q) + \mathbf{K}_q (\mathbf{I} - \mathbf{K})^{-1} (\mathbf{I} - \mathbf{K}^n) (\mathbf{T}(L) - \mathbf{T}(0)) \quad (4.23)$$

where $\mathbf{K}_q = e^{\mathbf{A}_{\kappa_q} \Delta t_q} \cdot e^{\mathbf{A}_{\kappa_{q-1}} \Delta t_{q-1}} \dots e^{\mathbf{A}_{\kappa_1} \Delta t_1}$. Equation (4.23) can be used to quickly calculate the temperature at $t = nL + t_q$, where $n \geq 1$ and $t_q \in [0, L]$. Moreover, let $n \rightarrow \infty$ in equation (4.23), we can quickly identify the steady-state temperature corresponding to t_q as

$$\mathbf{T}_{ss}(t_q) = \mathbf{T}(t_q) + \mathbf{K}_q(\mathbf{I} - \mathbf{K})^{-1}(\mathbf{T}(L) - \mathbf{T}(0)) \quad (4.24)$$

From equation (4.24), given a periodic schedule $\mathbf{S}(t)$, we can formulate the system steady-state temperature with information of the first period directly, which is much more efficient than to keep track of temperature variations based on equation (4.15).

4.4 Physical To Logical Mapping Heuristics

Before we introduce our mapping heuristics, we want to first guarantee that the timing constraints are met after re-mapping. We make one assumption that the highest frequency in $\mathbf{S}(t)$ can always be no greater than the maximum frequency supported by the core on which it is mapped. Under this assumption, we adjust the core's voltage-frequency mode such that the timing constraint satisfaction can be guaranteed. Specifically, we have two solutions for each interval if the current running mode cannot satisfy its nominal design parameter, 1) we change it to the next neighbor running mode or 2) we use the two neighboring speeds alternatively until the timing constraints are met.

Now we present three mapping approaches to solve Problem 6.2.1 as defined above. It is not difficult to prove that Problem 6.2.1 is in fact NP-hard. As mentioned before, while common approaches such as GA or SAA are commonly used to guide mapping decisions during the design phases, they are not applicable in *topology virtualization* approach due to their high timing complexities. Since mapping decisions must be made when installing or initiating the application, the key to the

success of this approach is the computation efficiency of the mapping algorithms and their effectiveness. In what follows, we develop three heuristics and study their effectiveness.

4.4.1 A Simple Utilization/Leakage Matching Heuristic

Our goal is to map the physical topology to the logical topology such that an existing nominal design can be improved in terms of peak temperature in the presence of core heterogeneity. As we discussed in Section 4.2.2, leakage variation is one of the key factors that affects temperature. An intuitive approach is therefore to match the logical core with the largest utilization to the least leaky physical core. The rationale behind this approach is that, when the larger utilization schedule is assigned to less leaky core, the less heat it generates. For example, consider two cores with identical voltage schedule (i.e., same utilization). The heat contributed by dynamic power is the same for both cores, but the one that is more leaky will generate more heat due to leakage power and therefore higher temperature. The algorithm is presented in Algorithm 4.

Algorithm 4 A simple heuristic to match high utilization logical core to low leaky physical core.

- 1: $\mathcal{M} = \emptyset$; // \mathcal{M} is the mapping solution space
 - 2: $LC =$ The sorted list of $\mathcal{A}_{i,j}^l \in A_{x \times y}^l, i = 0, \dots, x - 1; j = 0, \dots, y - 1$ by their utilizations based on nominal design in decreasing order;
 - 3: $LP =$ The sorted list of $\mathcal{A}_{i,j}^p \in A_{m \times n}^p$ by $\Delta_{leak}^{ij}, i = 0, \dots, m - 1; j = 0, \dots, n - 1$ in increasing order;
 - 4: **for** $i = 0$ to $sizeof(LC) - 1$ // for each logical core in the sorted list **do**
 - 5: **if** The utilization assigned to $LC(i) > 0$ **then**
 - 6: $\mathcal{M} = \mathcal{M} + \{LC(i) \rightarrow LP(i)\}$;
 - 7: **end if**
 - 8: **end for**
-

Algorithm 4 sorts the logical cores based on the assigned utilizations and the physical cores based on their leakage variations. Then, a physical core is mapped one by one to a logical core according to these two lists. The complexity of the algorithm mainly comes from sorting of the two lists. We assume the physical mesh is larger than the logical mesh. Therefore, the complexity of Algorithm 4 is $O((m \times n)\log(m \times n))$.

Algorithm 4 is fast and intuitive, but it has several issues. First, Algorithm 4 does not take the heat transfers from neighboring cores into account. In general, high utilization schedule can result in lower peak temperature when executed on a less leaky core. However, if several such cores are very close to each other, allocating high utilization schedules to these cores can result in high chip temperature. Second, utilization defined in this work is more related to the average power consumption. In fact, temperature varies more closely with power density rather than the average power consumption. In what follows, we develop two approaches to address these problems.

4.4.2 Hot-Cold Swapping

Given the *nominal design*, we can calculate the steady-state temperature for each core by the method we proposed in Section ??, based on which we can easily calculate the peak temperature when temperature reaches the stable status. By calculating the peak temperature, this method avoids the pitfall in the previous method to identify the peak temperature based on the schedule utilization. Then, *Hot-Cold Swapping* algorithm swaps the physical to logical topology between the hottest and coldest cores as shown in Algorithm 5. Similar to the principle for the

Algorithm 5 Hot-Cold swapping.

- 1: Initialize \mathcal{M} ; // by initial mapping algorithm or Algorithm 4
 - 2: **while** User defined stop criteria not satisfied **do**
 - 3: Calculate $\mathbf{T} = \{T_1, T_2, \dots, T_{m \times n}\}$, where T_i is the steady-state temperature of core i in \mathcal{M} ;
 - 4: $\mathcal{A}_{max}^l =$ The logical core with maximum temperature $T_{max} = \max(\mathbf{T})$;
 - 5: $\mathcal{A}_{min}^l =$ The logical core with minimum temperature $T_{min} = \min(\mathbf{T})$;
 - 6: //Swap the mapping between \mathcal{A}_{max}^l and \mathcal{A}_{min}^l
 - 7: $LC(\mathcal{A}_{max}^l) \rightarrow LP(\mathcal{A}_{min}^l)$;
 - 8: $LC(\mathcal{A}_{min}^l) \rightarrow LP(\mathcal{A}_{max}^l)$;
 - 9: Denote the new mapping as \mathcal{M}' ;
 - 10: Calculate \mathbf{T}' ; // steady-state temperature of new mapping \mathcal{M}'
 - 11: **end while**
-

“heat-and-run” heuristic [63], this method always exchanges the voltage schedules for the hottest/coldest cores, with the expectation that the heat across the chip can be spread and balanced in the entire chip until certain criteria (such as a pre-set peak temperature limit or loop counts) are reached. The complexity of Algorithm 5 is mainly from calculating the stable status temperature according to equation (4.24). Note that the dimension of matrix \mathbf{A}_{κ_q} is $(x \times y) \times (x \times y)$. Since the complexity for the straightforward implementation of the matrix multiplication and inversion are both $O(n^3)$ for $n \times n$ matrices, the complexity for each iteration in Algorithm 5 is $O(s \times (x \times y)^3)$ where s is the number of scheduling points for $\mathbf{S}(t)$.

While the *Hot-Cold Swapping* algorithm is simple, it does not necessarily reduce the peak temperature when swapping the schedules on a pair of hot/cold cores each time. The problem with this is that this approach does not take the heat transfers into consideration. Consider a core with light workload but surrounded with high workload cores and thus becomes the hottest core. When changing the schedule of it with other cold cores of lower temperature but higher workload, the peak temperature can become even higher.

4.4.3 Enhanced Hot-Cold Swapping

To solve the problem for *Hot-Cold Swapping*, we develop an *Enhanced Hot-Cold Swapping* as shown in Algorithm 6. The major difference between the two algorithms is that, in our *enhanced hot-cold swapping* algorithm, we tentatively swap the hottest and coldest cores. The swapping is accepted only when the new peak temperature is lower than the original one. If the peak temperature of the new mapping is higher than the initial mapping, we search for the core with second minimum temperature and swap it with the hottest core, until we can find such a swapping that reduces the peak temperature or we have exhausted all the possibilities. In the worst case, there are $(m \times n) - 1$ pairs of processor to be tested. Therefore, the complexity to run one iteration of Algorithm 6 is $O(s \times (m \times n) \times (x \times y)^3)$, where s is the number of scheduling points for $\mathbf{S}(t)$, $(m \times n)$ is the matrix size for the physical topology, and $(x \times y)$ is the matrix size for the logical topology. By ensuring the peak temperature non-increasing, *Enhanced Hot-Cold Swapping* heuristic can be more effective in guiding the search process to identify the good physical to logical topology mapping. In the next section, we use experiments to evaluate the performance of these algorithms.

4.5 Experimental Results

In this section, we perform three sets of experiments. First, we compare the peak temperature differences between with and without heterogeneity-aware approaches. Second, we study the performances of three approaches presented in Section ?? and compare our algorithms with *nominal design* and the optimal solution *exhaustive search* which enumerates all the possibilities. The last experiment is to compare the computation costs between different algorithms.

Algorithm 6 Enhanced Hot-Cold swapping.

```
1: Initialize  $\mathcal{M}$ ; // by initial mapping algorithm or Algorithm 4
2:  $\Lambda = 0$ ;
3: Calculate  $T = \{T_1, T_2, \dots, T_{\mathcal{P}}\}$ , where  $T_i$  is the steady-state temperature of core
    $i$  in  $\mathcal{M}$ ;
4:  $T^* = T$ ;
5: while  $\Lambda < \epsilon$  // user defined threshold do
6:    $\mathcal{A}_{max}^l =$  The logical core with maximum temperature  $T_{max}^* = \max\{T^*\}$ ;
7:    $\mathcal{A}_{min}^l =$  The logical core with minimum temperature  $T_{min}^* = \min\{T^*\}$ ;
8:   //Swap the mapping between  $\mathcal{A}_{max}^l$  and  $\mathcal{A}_{min}^l$ 
9:    $LC(\mathcal{A}_{max}^l) \rightarrow LP(\mathcal{A}_{min}^l)$ ;
10:   $LC(\mathcal{A}_{min}^l) \rightarrow LP(\mathcal{A}_{max}^l)$ ;
11:  Denote the new mapping as  $\mathcal{M}'$ ;
12:  Calculate  $T'$ ; // steady-state temperature of new mapping  $\mathcal{M}'$ 
13:  if ( $\max\{T'\} \geq \max\{T^*\}$ ) then
14:     $T^* = T^* - \{T_{min}^*\}$ ;
15:  else
16:    if ( $T^* == \{T_{max}^*\}$ ) then
17:      break;
18:    else
19:       $\mathcal{M} = \mathcal{M}'$ ;
20:       $\Lambda = \frac{\max\{T\} - \max\{T'\}}{\max\{T\}}$ ;
21:    end if
22:  end if
23: end while
```

4.5.1 Experimental Setup

In our simulation study, the multi-core platform consists a 2-D 3×3 mesh. We adopt the processor model from [111], each core supports 15 active modes with supply voltages ranging from 0.6V to 1.3V with step interval of 0.05V while the maximum frequency is generated as normal distribution and the frequency of each running mode is calculated accordingly [95]. For each core, we generate a static, periodic voltage schedule. Specifically, we divide the first period into 50 state intervals equally, for each state interval we randomly select one voltage mode from 0.6V to 1.3V. After generating the voltage schedule for each core within the first period,

we repeat the same schedule pattern for the rest of periods. As we discussed in Section 4.2.2, the running mode of each core is constant within each state interval.

We calculate the curve fitting parameters similar to the power model discussed in [133, 132] shown in Table I 4.0(a) and parameters from HotSpot-5.02 [149] shown in Table I 4.0(b) which we use for temperature calculation. Leakage variations are randomly generated as normal distribution: $\Delta_{leak} \sim N(\mu, \sigma)$, where $\mu = 0$ and $\sigma = 0.1 \times \bar{v}$ (\bar{v} is the average voltage speed of $V_{dd}(v)$ in Table I 4.0(a)) based on [24, 168]. The ambient temperature is $T_{amb} = 30^\circ C$.

All experiments are running on a Window XP/SP3 platform powered by Intel(R) Core(TM)2 Duo CPU @ 2.93GHz with 3.21 GB of RAM.

(a) Power/thermal parameters (b) HotSpot parameters

$V_{dd}(v)$	α	β	γ
0.00	0.0	0.0	0.0
0.60	0.2734	0.1313	16
0.65	0.5764	0.1383	16
0.70	0.9606	0.1457	16
0.75	1.4508	0.1534	16
0.80	2.0804	0.1615	16
0.85	2.8944	0.1700	16
0.90	3.9538	0.1789	16
0.95	5.3415	0.1882	16
1.00	7.1701	0.1979	16
1.05	9.5926	0.2081	16
1.10	12.8179	0.2188	16
1.15	17.1306	0.2300	16
1.20	22.9195	0.2416	16
1.25	30.7152	0.2538	16
1.30	41.2430	0.2665	16

Parameter	Value
Total Cores	9 (3×3)
Area per Core	4 mm ²
Die Thickness	0.15
Heat Spreader Side	20 mm
Heat Sink Side	30 mm
Convection Resistance	0.1 K/W
Convection Capacitance	140 J/K
Ambient Temperature	30° C

Table 4.1: Experiment parameters

4.5.2 Temperature vs. Leakage Variation

In this experiment, we first study the need to take leakage power consumption variations into consideration for temperature calculations. Utilization of each core

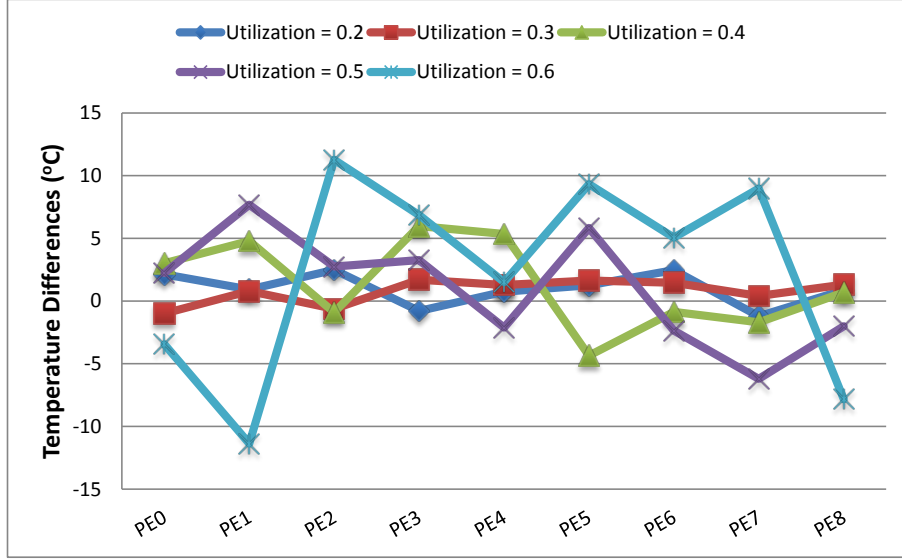


Figure 4.3: Temperature comparisons between different utilizations

U_i is calculated by equation (4.3). For different utilizations, we compare temperature differences for each core with and without leakage variations. The leakage variations are generated according to section 4.5.1 and each test case is running for 20 rounds.

In Figure 4.3, we compare temperature differences on 5 different utilization settings $U_i \in [0.2, 0.3, 0.4, 0.5, 0.6], i = 1, 2, \dots, \mathcal{P}$. X-axis represents core IDs while Y-axis represents temperature differences between the cases with and without leakage variations. As indicated from the figure, with the increase of utilization, the temperature variations for each core is also increasing. The larger the utilization is, the larger the discrepancy in temperature calculation. For example, the temperature difference is no more than $3^\circ C$ when utilization is 0.2 while the temperature difference can be as large as more than $10^\circ C$ when utilization is 0.6. Therefore, for a more accurate temperature calculation and peak temperature optimization, we need to take leakage variations into consideration.

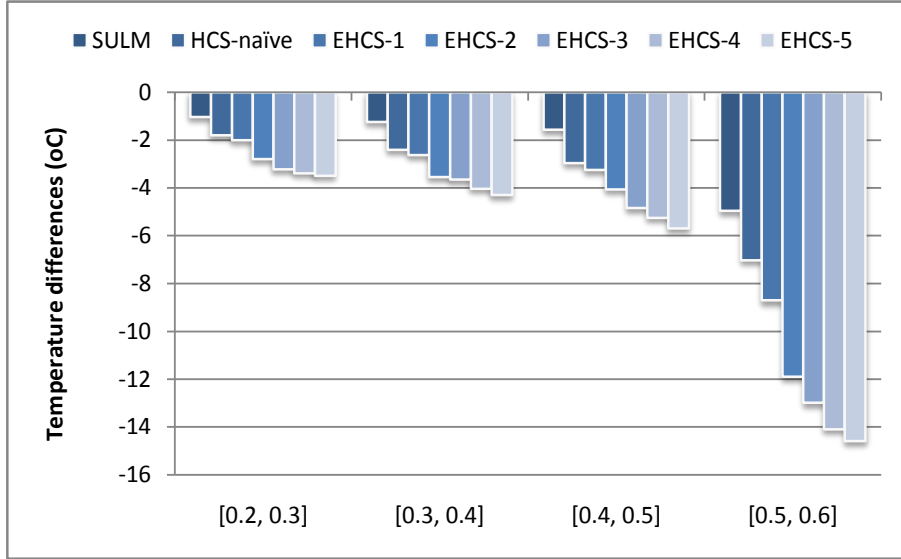


Figure 4.4: Peak temperature reductions normalized to initial mapping

4.5.3 Peak Temperature Minimization

Next, we study different approaches to optimize peak temperature. We randomly generate voltage schedule for each core with utilization and they are labeled as *nominal design* ($U_i \in \{[0.2, 0.3], [0.3, 0.4], [0.4, 0.5], [0.5, 0.6]\}, i = 1, 2, \dots, \mathcal{P}$). The reason we limit core’s utilization to $U_i = 0.6$ is to satisfy the peak temperature constraint $T_{max} = 95^\circ C$ in steady state ($T_{max} = 95^\circ C$ is the temperature threshold we choose). We assume that the logical architecture and physical architecture are of equal size, i.e., $x = m$ and $y = n$. When the physical topology size is larger than the logical topology size, it is not difficult to see that our heuristics can perform better simply because of the extra resources or optimization opportunities available. For each utilization setting, we generate 100 test cases and calculate the average temperature reduction.

We denote our three optimization methods as *SULM* (the *simple utilization/leakage matching* heuristic), *HCS-naïve* (the *Hot-Cold Swapping* heuristic) and *EHCS* (the *Enhanced Hot-Cold Swapping* heuristic). If no temperature reduction can be made,

EHCS will stop. *EHCS-1* refers to our *Enhanced Hot-Cold swapping* algorithm with one iteration.

In Figure 4.4, we compare peak temperature reductions between *SULM*, *HCS-naive*, and *EHCS-1* to *EHCS-5* with 4 different utilization settings ranging from $[0.2, 0.3]$ to $[0.5, 0.6]$. From the figure, the first conclusion we can make is that with the increase of utilization, all heuristics can get more temperature reductions. It is because the higher utilization we have, the more potential we may benefit from heterogeneity-aware algorithms which try to avoid putting higher utilizations on more leaky cores. Second, *SULM* performs the worst because it does not take heat transfers between neighbors into account. *HCS-naive* is better than *SULM* because it does consider heat transfers, but compared to *EHCS*, *HCS-naive* would always swap the physical to logical topology between the hottest and coldest cores regardless and it is possible that sometimes the peak temperature after swapping is higher than the *nominal design*, therefore hampering its performance. From *EHCS-1* to *EHCS-5*, it indicates that the more iterations we run the better peak temperature reduction we can get.

One thing we need to note is that with utilization setting of $[0.5, 0.6]$, *EHCS* can perform much better than previous settings. Higher utilization indicates that each core has larger possibility to run at high voltage modes that are more sensitive to core heterogeneity. Therefore, with heat transfers and leakage variations into account, *EHCS* can benefit more than *SULM* and *HCS-naive* from increasing utilization.

Another set of experiments is shown in Figure 4.5. We want to see how good *EHCS* algorithm is compared to the optimal solution *exhaustive search* which enumerates all the mapping possibilities. This time we generate each core's utilization

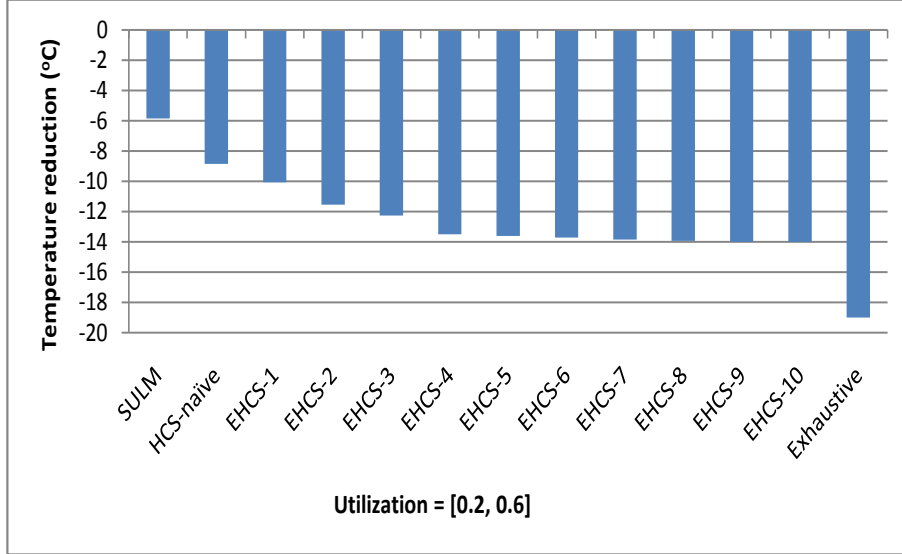


Figure 4.5: Temperature reductions between *EHCS* and Exhaustive search

$U_i \in [0.2, 0.6], i = 1, 2, \dots, \mathcal{P}$ for a more general purpose. We perform 100 test cases for each algorithm and calculate the average.

From the figure, *SULM* can get $5.86^\circ C$ reduction, *HCS-naïve* can get $8.79^\circ C$ reduction. From *EHCS-1* to *EHCS-10* can get $10.16^\circ C$ all the way to $14.09^\circ C$ reduction, while exhaustive search can get $18.99^\circ C$ reduction in average. Note that after *EHCS-5* the reduction potential is not significant. Therefore, we can choose different iterations based on the timing and improvement we want to achieve. In general, compared with *exhaustive search*, *EHCS-5* algorithm only performs less than $5^\circ C$ of difference.

4.5.4 Operational Costs

As mentioned earlier, the computation efficiency plays a vital role in *topology virtualization*. We next study the computational cost for different approaches with utilization setting $U_i \in [0.2, 0.6], i = 1, 2, \dots, \mathcal{P}$. For simplicity, we just compare

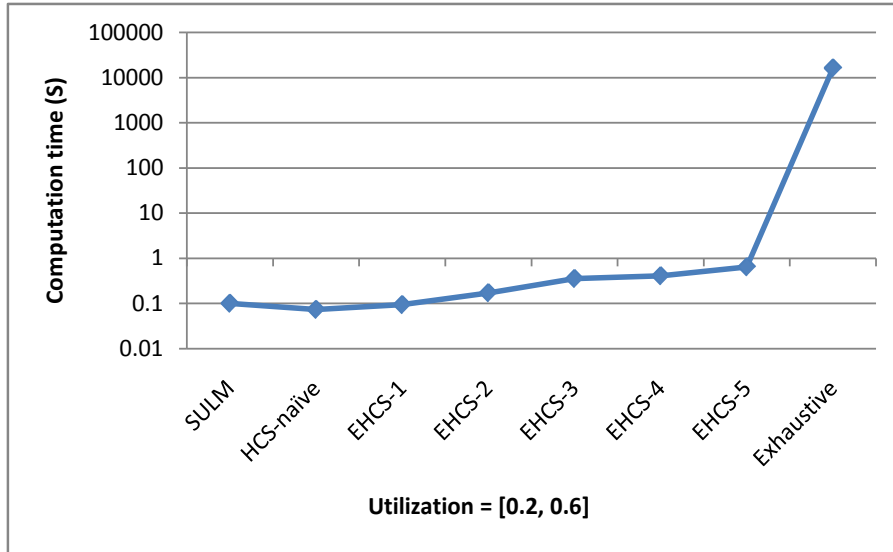


Figure 4.6: Computation time differences between different approaches

from *EHCS-1* to *EHCS-5* to give a trend of what the timing complexity looks like. Figure 4.6 shows the time that each algorithm takes when running only one test case, while *EHCS-1* to *EHCS-5* with different iterations from one to five. From the figure, it is indicated that the timing complexity is linearized and it is what we expected since we need such a mapping heuristic that can perform fast and produce relatively good results. *EHCS-5*, which is the most time consuming among the five, takes about 0.64 seconds to finish. However, *exhaustive search* is very time consuming, the computation complexity is $O(P!)$ which takes 4 to 5 hours to finish in our experiment.

4.6 Summary

The temperature minimization problem is becoming more and more critical in computer system design. In the meantime, the increasing process variation for IC chips also raise the concerns in the design of computing systems. We believe that the het-

erogeneity caused by process variation, if utilized appropriately, can in fact improve the design objectives of real-time applications. In this work, we develop three heuristics to judiciously mirror the underlying physical architecture of an individual device to the logical architecture with the objective of peak temperature minimization. The proposed heuristics can achieve $14.09^{\circ}C$ temperature reduction in average and less than $5^{\circ}C$ of difference compared with *exhaustive search*. Overall, our proposed algorithm can be finished within 1 second (more than 10^4 times faster compared to *exhaustive search*) which is the key to the success of optimization problems through *topology virtualization*.

CHAPTER 5

MULTI-CORE FIXED-PRIORITY SCHEDULING OF REAL-TIME TASKS WITH STATISTICAL DEADLINE GUARANTEE

In the previous chapter, we presented a leakage-aware task re-mapping algorithm to optimize the overall peak temperature on a multi-core platform. In this chapter, we attack real-time scheduling on multi-core platforms under uncertainty of tasks' execution times. Specifically, we adopt a probabilistic approach for fixed-priority preemptive scheduling of real-time tasks on multi-core platforms with statistical deadline miss ratio guarantee. Rather than a single-valued worst-case execution time (WCET), we formulate the task execution time as a probabilistic distribution. We develop a novel algorithm to partition real-time tasks on multiple homogenous cores, which takes not only task execution time distributions but their period relationships into considerations. Our extensive experimental results show that our proposed methods can greatly improve the schedulability of real-time tasks when compared with the traditional bin packing approaches.

5.1 Related Work

The traditional real-time system analysis adopts a deterministic approach, i.e. based on deterministic real-time parameters such as the worst-case execution times (WCET), and provides a deterministic guarantee [114, 104, 103] such that all jobs from every single task can meet their deadlines. As the computing performance becomes less and less predictable, such a deterministic design can lead to extremely pessimistic design. In addition, the hard deadline guarantee may not be necessary for many soft real-time systems that allow a portion of the jobs miss their deadlines. For example, for aerospace industry, a probability of failure of 10^{-15} per hour is considered to be

feasible compared to the maximum allowed probability of failure of 10^{-9} per hour that is required by the certification authorities [2].

The probabilistic approach takes system probabilistic characteristics, such as the execution times, into account for real-time system analysis and design to prevent over-provisioning and, at the same time, meet real-time constraints [43]. There have been increasing interests from real-time community on probabilistic approaches for real-time system analysis and design. For example, Tia et al. [158] presented a probabilistic performance guarantee for semi-periodic tasks by transforming semi-periodic tasks into a periodic task followed by a sporadic task. Atlas et al. [14] introduced statistical rate monotonic scheduling for periodic tasks with statistical QoS requirements. Maxim et al. [119] proposed three priority assignment algorithms for probabilistic real-time systems. They further improved the previous work by proposing a framework of re-sampling mechanism that can simplify the response time distributions in order to ease timing analysis for real-time systems in [121]. Yue et al. [117] presented a statistical response time analysis by analyzing samples in timing traces taken from real systems. In [91], the authors proposed a stochastic analysis framework which computed the response time distribution and deadline miss probability for each individual task. The framework can be applied to both fixed-priority and dynamic-priority systems on a single-core platform. The authors in [120] extended their work to allow both task's execution time and period to be random variables and computed analytically the response time distribution of the tasks on uniprocessor under a task-level fixed-priority preemptive scheduling policy. In [16], the authors proposed a new convolution-based stochastic analysis in which they modeled faults as additional execution time to bound the probability to exceed a response-time value in the worst-case on single processor under fixed-priority non-preemptive scheduling policy.

In this work, we are interested in the problem of how to schedule a set of fixed-priority real-time tasks with probabilistic execution times on multiple homogeneous processing cores and satisfy the given deadline miss probabilities. We focus on fixed-priority scheduling schemes since fixed-priority scheduling is one of the most popular scheduling schemes in real-time system design. It has simpler implementation and better practicability than other dynamic priority-based schedulings [20]. Given the NP-hard nature of this problem [54], one intuitive approach is to transform the problem into a simple bin-packing problem [84], and employ the feasibility test methods developed in [91] to ensure the deadline miss probability guarantee.

Note that, it is a well known fact that the period relationship among tasks, if exploited appropriately, can greatly improve the processor utilization [47, 48]. The challenge however is how to determine if a task is more “harmonic” than another one to a reference task if their periods are not strictly integer multiples, and their execution times are probabilistic instead of deterministic. To this end, we develop four novel metrics, with one improving upon another, to quantify the degree of harmonicity between two tasks. Based on these metrics, we then develop an algorithm that takes both the probabilistic execution times and task period relationship into consideration to guide the partition process for periodic tasks with random execution times on multi-core platforms. We have conducted extensive simulations to validate our approach. The experimental results show that the proposed approach can significantly improve the schedulability of real-time tasks when compared with traditional bin-packing approaches.

The rest of the chapter is organized as follows. In Section 5.2 we introduce our system models and formally define our problem. In section 5.3 we present the harmonic-aware metrics we developed. In Section 5.4 we talk about our task

partition algorithm in detail. Section 5.5 presents the experimental results and finally, we conclude in Section 5.6.

5.2 Preliminary

In this section we first introduce our system models such as task models and processor models, and then we formulate the problem formally.

5.2.1 System Models And Problem Formulation

We consider a real-time system consisting of N independent periodic tasks, denoted as $\Gamma = \{\tau_1, \tau_2, \dots, \tau_N\}$, to be scheduled on a homogeneous multi-core platform, denoted as $\mathcal{P} = \{p_1, p_2, \dots, p_K\}$, according to the Rate Monotonic Scheduling (RMS) policy. Each task $\tau_i \in \Gamma$, characterized by a tuple (\mathcal{C}_i, T_i) , where

$$\mathcal{C}_i = \begin{pmatrix} c_1 = c_{min} & \dots c_k & \dots & c_n = c_{max} \\ Pr(c_{min}) & \dots Pr(c_k) & \dots & Pr(c_{max}) \end{pmatrix} \quad (5.1)$$

representing the execution time distribution of τ_i . That is, the probability that the execution time of $\mathcal{C}_i = c_k$ is $Pr(c_k)$. For all possible values of \mathcal{C}_i , we have $c_k \in [c_{min}, c_{max}]$, where c_{min} and c_{max} are the minimum and maximum values for \mathcal{C}_i , and $\sum_{k=1}^n Pr(c_k) = 1$. T_i is the period of task τ_i which is a constant value. We assume deadline equals to period in this work, $D_i = T_i$. Since a task's execution time is not unique, the response time for each job may be different. Therefore sometimes a job may meet or miss its deadline. We formally define the concept of *deadline miss probability* as follows.

Definition 5.2.1. *The deadline miss probability (DMP) of job $\tau_{i,j}$ (denoted as $DMP_{i,j}$) is the probability that job $\tau_{i,j}$ misses its deadline and can be formulated as following:*

$$DMP_{i,j} = Pr(\mathcal{R}_{i,j} > D_{i,j}) \quad (5.2)$$

where $\mathcal{R}_{i,j}$ is the response time distribution of job $\tau_{i,j}$ and $D_{i,j}$ is the deadline of job $\tau_{i,j}$. Accordingly, the deadline miss probability of task τ_i (denoted as DMP_i) can be formulated as

$$DMP_i = \max\{DMP_{i,j}\}, \tau_{i,j} \in \tau_i, \quad (5.3)$$

and the deadline miss probability for a task set Γ (denoted as DMP_Γ) can be formulated as

$$DMP_\Gamma = \max\{DMP_i\}, \tau_i \in \Gamma. \quad (5.4)$$

Our research problem can be formulated as follows:

Problem 5.2.2. *Given*

- *a task set consisting of N tasks, $\Gamma = \{\tau_1, \tau_2, \dots, \tau_N\}$,*
- *a multicore platform with K homogeneous processing cores, $\mathcal{P} = \{p_1, p_2, \dots, p_K\}$,*
- *and the deadline miss probability, DMP_Γ ,*

partition the task set Γ on the multi-core platform and schedule the tasks on each core using RMS scheme such that the deadline miss probability constraint of the task set is satisfied and the number of cores is minimized.

5.2.2 Motivations

One simple approach for this problem is to transform it into the traditional bin-packing problem. Note that, with the knowledge of tasks assigned to a processing

core, Lopez et al. [116] proposed a method to calculate the probabilistic response time of a job under a preemptive uniprocessor fixed-priority scheduling policy, which can be further applied to determine if the DMP constraints can be satisfied. Therefore, traditional bin-packing approaches such as First Fit, Next Fit, Best Fit can be applied to assign tasks to different cores.

It is a well known fact that, for RMS, the processor utilization can reach as high as one if the tasks are harmonic, i.e. task periods are integer multiples of each other. For tasks that are not entirely harmonic, Fan et al. [47] showed that, if the period relationships among tasks can be appropriately exploited, the processor utilization can be significantly improved. Specifically, they introduce three interesting concepts, *sub harmonic task set*, *the primary harmonic task set* and *harmonic index*, which are defined as follows:

Definition 5.2.3. [47] Given a task set $\Gamma = \{\tau_1, \tau_2, \dots, \tau_N\}$, let $\hat{\Gamma} = \{\hat{\tau}_1, \hat{\tau}_2, \dots, \hat{\tau}_N\}$, where $\hat{\tau}_i = (\mathcal{C}_i, \hat{T}_i)$, $\hat{T}_i \leq T_i$, and $\hat{T}_i | \hat{T}_j$ if $T_i < T_j$ ($a|b$ means "a divides b" or "b is an integer multiple of a"). Then $\hat{\Gamma}$ is a sub harmonic task set of Γ .

Definition 5.2.4. [47] Let Γ' be a sub harmonic task set of Γ . Then Γ' is called a primary harmonic task set of Γ if there exists no other sub harmonic task set Γ'' such that $T'_i \leq T''_i$ for all $1 \leq i \leq N$.

Definition 5.2.5. [47] Given a task set Γ , let $\mathcal{G}(\Gamma)$ represent all the primary harmonic task sets of Γ . Then the harmonic index of Γ , denoted as $\mathcal{H}(\Gamma)$, is defined as

$$\mathcal{H}(\Gamma) = \min_{\Gamma' \in \mathcal{G}(\Gamma)} \Delta(U') \quad (5.5)$$

where

$$\Delta(U') = \begin{cases} U(\Gamma') - U(\Gamma) & \text{if } U(\Gamma') \leq 1, \\ +\infty & \text{otherwise.} \end{cases} \quad (5.6)$$

$U(\Gamma)$ and $U(\Gamma')$ represent the utilizations of task set Γ and Γ' , respectively. We can employ *Sr* or *DCT* algorithm [69, 70] to find all the sub-harmonic task sets with a complexity as low as $N \cdot \log(N)$.

When allocating tasks to processors, they either allocate a task to a processor that can minimize the harmonic index, or pick the sub-task sets with highest degree of harmonic and assign them to a processor.

We believe that, by exploiting the period relationships among tasks, we can also greatly improve the processor utilization in Problem 6.2.1. The challenge is how to quantify the degree of harmonic among different tasks with probabilistic execution times. For tasks with deterministic execution times, according to Definition 5.2.5, for a given reference task, a task with its original period closer to the transformed period in its primary harmonic task set has a higher degree of harmonicity. However, the degree of harmonicity of a task to its reference task may depend not only on its period but its execution time distribution as well. Consider a task set with three tasks $\tau_a = \left\{ \left(\begin{smallmatrix} 2 & 3 \\ 0.3 & 0.7 \end{smallmatrix} \right), 6 \right\}$, $\tau_b = \left\{ \left(\begin{smallmatrix} 4 & 6 \\ 0.5 & 0.5 \end{smallmatrix} \right), 12 \right\}$, and $\tau_c = \left\{ \left(\begin{smallmatrix} 3 & 7 \\ 0.5 & 0.5 \end{smallmatrix} \right), 12 \right\}$. Note that both τ_b and τ_c have the same period. If we combine τ_a and τ_b , we have $DMP_{\tau_a, \tau_b} = 0$. If we combine τ_a and τ_c , we have $DMP_{\tau_a, \tau_c} = 24.5\%$. Therefore, the degree of harmonicity of a task depends not only on its period, but its execution time distribution as well.

In what follows, we first introduce four metrics that we have developed, with each one improving upon the previous, to quantify the degree of harmonicity between two tasks. We then propose an algorithm that takes both the probabilistic execution times and task period relationships into consideration to guide the partition process for periodic tasks with random execution times on multi-core platforms.

5.3 Harmonic Index For Tasks With Probabilistic Execution Times

In this section, we formally introduce the metrics which take harmonic relationships into consideration to guide our allocation of tasks with random execution times. Since not all tasks in a task set are strictly harmonic, it is desirable that we quantify the *harmonic* of tasks and allocate tasks with high degree of harmonicity to the same processor to achieve high utilization as well as high schedulability. In what follows, we develop four metrics to measure the task harmonic relationships.

5.3.1 Mean-Based Harmonic Index

Our goal is to quantify the degree of harmonicity between two tasks. Before we define the harmonic index for this purpose, we first introduce the following concept.

Definition 5.3.1. *Given a task set $\Gamma = \{\tau_1, \tau_2, \dots, \tau_N\}$ and one of its primary harmonic task set $\Gamma' = \{\tau'_1, \tau'_2, \dots, \tau'_N\}$, let $\tau_r \in \Gamma$, $\tau'_r \in \Gamma'$ and $\tau_r = \tau'_r$. Then τ_r is called the reference task of the primary harmonic task set Γ' , and Γ' is called the primary harmonic task set based on τ_r and is also denoted as $\Gamma'(\tau_r)$.*

According to Definition 5.3.1, the primary harmonic task set based on τ_r , i.e. $\Gamma'(\tau_r)$, is simply the primary harmonic task set with τ_r unchanged.

When task execution times are probabilistic, one intuitive approach is to employ the execution time mean and thus transform the probabilistic execution time distribution into a deterministic value. The harmonic index can be therefore defined in a similar way as that for tasks with deterministic execution times.

Definition 5.3.2. *Given a task $\tau_i = \{\mathcal{C}_i, T_i\} \in \Gamma$ and its reference task $\tau_r \in \Gamma$, let $\tau'_i = \{\mathcal{C}_i, T'_i\}$ be the corresponding task of τ_i in $\Gamma'(\tau_r)$. Then the Mean based*

harmonic index of task τ_i w.r.t. the reference task τ_r , denoted as $\mathcal{H}_m(\tau_i, \tau_r)$, is defined as

$$\mathcal{H}_m(\tau_i, \tau_r) = |\bar{U}(\tau_i) - \bar{U}(\tau'_i)|, \quad (5.7)$$

where

$$\bar{c}_i = \sum_{\forall (c_k, Pr(c_k)) \in \mathcal{C}_i} c_k \cdot Pr(c_k), \quad (5.8)$$

$$\bar{U}(\tau_i) = \frac{\bar{c}_i}{T_i}. \quad (5.9)$$

Note that τ_r in Equation 5.7 indicates $\bar{U}(\tau'_i)$ is calculated under its corresponding task set $\Gamma'(\tau_r)$.

Table 5.1: Sub harmonic task set transformations of a 4-task set

τ_i	(\mathcal{C}_i, Pr_i)	T_i	Transformed based on τ_1		Transformed based on τ_2	
			\hat{T}_i	$\mathcal{H}_m(\tau_i, \tau_1)$	\hat{T}_i	$\mathcal{H}_m(\tau_i, \tau_2)$
1	(2, 0.3) (3, 0.7)	6	6	0	5	0.09
2	(4, 0.5) (5, 0.5)	10	6	0.3	10	0
3	(4, 0.5) (6, 0.5)	12	12	0	10	0.083
4	(8, 0.7) (10, 0.3)	20	18	0.032	20	0

Let us consider the example shown in Table 5.1. A task set contains four independent periodic tasks, each with a probabilistic execution time distribution and a deterministic period. We transform the original task set into two primary harmonic task sets, based on τ_1 and τ_2 , respectively (for more details, please check [70]). For the first primary harmonic task set which is transformed based on task τ_1 ,

we take τ_2 as an example to show how we derive its mean-based harmonic index $\mathcal{H}_m(\tau_2, \tau_1)$. According to Equation 5.7, $\bar{U}(\tau_2) = 0.45$ and $\bar{U}(\tau'_2) = 0.75$. Therefore, $\mathcal{H}_m(\tau_2, \tau_1) = |\bar{U}(\tau_2) - \bar{U}(\tau'_2)| = 0.3$. Then if we sort the tasks based on $\mathcal{H}_m(\tau_i, \tau_1)$, we have $\mathcal{H}_m(\tau_1, \tau_1) = 0$, $\mathcal{H}_m(\tau_3, \tau_1) = 0$, $\mathcal{H}_m(\tau_4, \tau_1) = 0.032$ and $\mathcal{H}_m(\tau_2, \tau_1) = 0.3$. Then we tentatively combine task τ_1 with τ_3 , task τ_1 with τ_4 and task τ_1 with τ_2 into a sub set, to check deadline miss probability of each individual subset. We can get: $DMP_{\tau_1, \tau_3} = 0\%$, $DMP_{\tau_1, \tau_4} = 19.55\%$ and $DMP_{\tau_1, \tau_2} = 24.5\%$. It shows that smaller \mathcal{H}_m does imply better harmonic relationship between two tasks.

From Definition 5.3.2, we can see that the mean based harmonic index (\mathcal{H}_m) quantifies the harmonic relationship of a task to its reference task by measuring the difference of its expected utilization with that in the primary harmonic task set. While mean value is a good representative value for a probabilistic distribution, it cannot capture the entire characteristics of a probabilistic distribution. Let us recall the example shown in Sub-section 6.2.2; task τ_b and τ_c not only have the same period but also the same mean. According to \mathcal{H}_m , the two tasks have the same harmonic index. However, the scheduling results are different ($DMP_{\tau_a, \tau_b} \neq DMP_{\tau_a, \tau_c}$). More effective harmonic index needs to be developed.

5.3.2 Variance-Based Harmonic Index

From the example in the previous sub-section, we can observe that the harmonic index depends not only on the mean value of the execution times, but also on the variance of the execution times. It is therefore reasonable to take the variance into consideration when designing the harmonic metric. To this end, we develop a variance-based harmonic index as follows.

Definition 5.3.3. Given $\tau_i \in \Gamma$ and its reference task $\tau_r \in \Gamma$, let $\tau'_i = \{\mathcal{C}_i, T'_i\}$ be the corresponding task of τ_i in $\Gamma'(\tau_r)$. Then the Variance-based harmonic index of task τ_i w.r.t. the reference task τ_r , denoted as $\mathcal{H}_v(\tau_i, \tau_r)$, is defined as

$$\mathcal{H}_v(\tau_i, \tau_r) = \mathcal{H}_m(\tau_i, \tau_r) + Var(\tau_i, \tau_r), \quad (5.10)$$

where

$$Var(\tau_i, \tau_r) = \frac{\sqrt{\sum_{\forall c_k \in \mathcal{C}_i} (c_k - \bar{c}_i)^2 \cdot Pr(c_k)}}{T'_i} \quad (5.11)$$

\mathcal{H}_v improves upon \mathcal{H}_m by taking both the mean value of execution times as well as their variance into considerations. For the example shown in Sub-section 6.2.2, we have $\mathcal{H}_v(\tau_a, \tau_b) < \mathcal{H}_v(\tau_a, \tau_c)$ (since $\mathcal{H}_m(\tau_a, \tau_b) = \mathcal{H}_m(\tau_a, \tau_c)$ and $Var(\tau_a, \tau_b) < Var(\tau_a, \tau_c)$) indicating that task τ_b is more harmonic than τ_c to reference task τ_a . This conforms to the results from the schedulability analysis. However, there are still problems with the proposed harmonic metric. First, it essentially implies that both execution time mean values and variances are equally important in evaluating the degree of harmonic. Second, again, using only mean value and its variance cannot capture accurately the characteristics of execution time distributions. Many execution time distributions may have the same mean value and variance but totally different probabilistic characteristics.

5.3.3 Harmonic Index Based on Cumulative Distribution Function

We believe that we can achieve a better correlation of harmonic index and task schedulability if we can capture execution time distributions more accurately and incorporate them into the harmonic index. To this end, we propose another metric developed on the cumulative distribution function of task execution times. Before

we present our new harmonic index, we first introduce the following concepts and notations.

Definition 5.3.4. Given $\tau_i \in \Gamma$, the cumulative distribution function of the task's utilization, denoted as $CDF_{\tau_i}(x)$, can be formulated as

$$CDF_{\tau_i}(x) = \frac{Pr(\mathcal{C}_i \leq x)}{T_i} \quad (5.12)$$

Essentially, the cumulative distribution function is the utilization CDF of task τ_i . Note that CDFs for $\tau_i \in \Gamma$ and its corresponding task $\tau'_i \in \Gamma$ are different. To measure the "distance", we can use the ℓ^2 -norm operation.

Definition 5.3.5. Given a vector $x = [x_1, x_2, \dots, x_n]$, its ℓ^2 -norm, denoted as $\|x\|$, is defined as

$$\|x\| = \sqrt{\sum_{k=1}^n |x_k|^2} \quad (5.13)$$

Now we are ready to define the new harmonic index.

Definition 5.3.6. Given $\tau_i \in \Gamma$ and its reference task $\tau_r \in \Gamma$, let $\tau'_i = \{\mathcal{C}_i, T'_i\}$ be the corresponding task of τ_i in $\Gamma'(\tau_r)$. Then the Cumulative distribution function harmonic index of task τ_i to τ_r , denoted as $\mathcal{H}_C(\tau_i, \tau_r)$, is defined as

$$\mathcal{H}_C(\tau_i, \tau_r) = \|CDF_{\tau_i}(x) - CDF'_{\tau_i}(x)\| \quad (5.14)$$

where x represents sampling point for the two cumulative distribution functions, so that we can apply the above equation to measure harmonic index.

One thing to note is that for $CDF_{\tau_i}(x)$ and $CDF'_{\tau_i}(x)$, they may have different sampling points. For this case, we enumerate all the sampling points for both CDFs to calculate $\mathcal{H}_C(\tau_i, \tau_r)$.

The rationale behind the definition of *Cumulative distribution function harmonic index* is that we intend to determine the degree of harmonic by measuring how

much the task utilization distribution has changed after changing its period to be an integer multiple of the reference task. The larger the difference, the less harmonic the two tasks are.

As indicated in the equation, the *harmonic index* tries to evaluate the closeness between two distributions by summing up the square difference of each sampling point of the two distributions (CDFs) and then takes the root value as the final result. Consider the same example in Table 5.1. After all the tasks have been transformed based on the *reference task* τ_1 's period, we calculate the *harmonic index* between τ_i and τ_1 , i.e., $\mathcal{H}_C(\tau_i, \tau_1)$. In this way, we can rank the harmonic relationships of all the tasks compared to the *reference task* τ_1 and find out which tasks are more harmonic to task τ_1 .

5.3.4 The Utilization Sum Based Harmonic Index

Note that \mathcal{H}_C determines if τ_i is harmonic to τ_r only by the “distance” of utilization distributions of task τ_i in Γ and $\Gamma'(\tau_r)$. While the utilization of τ_i can affect the task schedulability, the combined utilization distribution of τ_i and τ_r can be a better indicator to the schedulability for task sets containing both τ_i and τ_r . Therefore, to design a harmonic index that can be a more effective schedulability indicator, it is reasonable to use the sum of utilization of both τ_i and τ_r rather than that of τ_i individually.

For ease of our presentation, we first introduce the following notation. Let $\tau_i \otimes \tau_r$ denote the convolution of \mathcal{C}_{τ_i} and \mathcal{C}_{τ_r} .

Definition 5.3.7. *Given a task τ_i and a reference task τ_r , let CDFs for $\tau_i \otimes \tau_r$ and $\hat{\tau}_i \otimes \tau_r$ be CDF_{τ_i, τ_r} and $CDF_{\hat{\tau}_i, \tau_r}$. Then the Utilization-sum-based harmonic index of $\tau_i \otimes \tau_r$ and $\hat{\tau}_i \otimes \tau_r$, denoted as $\mathcal{H}_S(\tau_i, \tau_r)$, is formulated in Equation 5.15,*

$$\mathcal{H}_S(\tau_i, \tau_r) = \|CDF_{\tau_i, \tau_r}(x) - CDF_{\hat{\tau}_i, \tau_r}(x)\| \quad (5.15)$$

\mathcal{H}_S can take the information of combined utilization distribution CDF_{τ_i, τ_r} between Γ and $\Gamma'(\tau_r)$ into account. Therefore it can produce better results in terms of harmonic relationships.

So far we have introduced all four metrics with each improving upon another. These metrics are critical for our task partition algorithm to make mapping decisions. In what follows, we present our task partition algorithm in details.

5.4 Task Partitioning Algorithm

With the harmonic indexes defined above, we are ready to introduce our task partitioning algorithm. Essentially, our algorithm intends to identify the tasks with highest harmonic index values, and put them into one processor to improve the processor utilization. To satisfy the DMP requirement, we conduct the schedulability analysis based on the technique proposed in [91]. The detailed algorithm is illustrated in Algorithm 7.

As shown in Algorithm 7, our algorithm chooses the reference task from the first task τ_1 until the last task τ_N . For each reference task, the rest of the tasks are ordered according to the chosen harmonic index values, and selected from high value to low to form a sub-task set until the *DMP* test is failed. After we identify all the subsets from each primary harmonic task set, we choose the best subset and allocate it to a processor. Then we delete these tasks from task set Γ . We repeat this process for the rest of the tasks until all tasks are assigned.

We want to explain how we choose the best subset by an example. We are still going to analyze the example shown in Table 5.1. Let us take task τ_1 as the reference task, then the two corresponding sub-sets: (τ_1, τ_3) and (τ_2, τ_4) , respectively, are both perfectly schedulable. Now the question is which one should we pick first in order to

generate a better subset? The criteria here is that we want to utilize the processors as much as possible, therefore, we would like to pick the subset with large utilization. However, since the tasks we are discussing in this work have random execution times, how should we define which subset has larger utilization than another?

From the task’s execution time distribution, we can easily get each task’s utilization distribution. Let \mathcal{U}_i denote the utilization distribution for task τ_i . For the tasks in Table 5.1 we have $\mathcal{U}_1 = \left(\begin{smallmatrix} \frac{2}{6} & \frac{3}{6} \\ 0.3 & 0.7 \end{smallmatrix} \right)$, $\mathcal{U}_2 = \left(\begin{smallmatrix} \frac{4}{10} & \frac{5}{10} \\ 0.5 & 0.5 \end{smallmatrix} \right)$, $\mathcal{U}_3 = \left(\begin{smallmatrix} \frac{4}{12} & \frac{6}{12} \\ 0.5 & 0.5 \end{smallmatrix} \right)$, and $\mathcal{U}_4 = \left(\begin{smallmatrix} \frac{8}{20} & \frac{10}{20} \\ 0.7 & 0.3 \end{smallmatrix} \right)$. The utilization distribution of a subset is the convolution of each task within the subset. So we have $\mathcal{U}_{1,3} = \mathcal{U}_1 \otimes \mathcal{U}_3 = \left(\begin{smallmatrix} 0.67 & 0.83 & 1.0 \\ 0.15 & 0.5 & 0.35 \end{smallmatrix} \right)$, similarly we have $\mathcal{U}_{2,4} = \left(\begin{smallmatrix} 0.8 & 0.9 & 1.0 \\ 0.35 & 0.5 & 0.135 \end{smallmatrix} \right)$ (transferred to decimal for better illustration). If we choose the utilization threshold as 0.8, we can calculate that the probability that subset (τ_1, τ_3) has utilization distribution greater than 0.8 is $Pr(\mathcal{U}_{1,3} > 0.8) = 0.85$ while that for subset (τ_2, τ_4) ’s under the same situation is $Pr(\mathcal{U}_{2,4} > 0.8) = 1.0$. So we will choose subset (τ_2, τ_4) first because it has higher probability to have larger utilization than subset (τ_1, τ_3) . The threshold we choose in this work is 0.7 (higher threshold may result in a small improvement of partitioning but more computational expense).

5.5 Experimental Results

In this section, we use experiments to investigate the effectiveness of our proposed algorithm. Three sets of experiments are conducted. First, we compared the performance in terms of number of cores necessary with different partitioning algorithms. Second, we compared the success ratios by different approaches when scheduling real-time tasks on a multi-core platform with a pre-defined core number. Finally, we compare the computational costs of approaches with different harmonic indexes for 8 tasks, 16 tasks, and 24 tasks, respectively.

5.5.1 Experiment Setup

In our experiments, we randomly generated the stochastic tasks. Specifically, for each task we generated four different possible execution times and the corresponding probabilities (the sum of the four probabilities equals to 1). We did not make any assumption regarding task’s execution time distribution, therefore, any distribution can be applied. We generated the periods for all the tasks in a way that the expected utilization of each task is evenly distributed within $[0.2, 0.5]$.

Five different approaches were realized in our experiment, i.e., four task partitioning algorithms with the four proposed harmonic indexes and one traditional bin packing approach, first fit.

We denote our approach using *mean-based harmonic index* as H_m , using *variance-based harmonic index* as H_v , the one using *cumulative-distribution-based harmonic index* as H_{cdf} and the last one using *distribution-sum-based harmonic index* as H_{sum} . Then we compare the four approaches with first fit (FF) algorithm.

Specifically, for FF approach, we sort the tasks according to their expected utilizations and pack as many tasks as possible from the top of the task queue one by one, until we can form a subset while meeting DMP_{Γ} constraint. After we successfully find a subset, we delete those tasks from the task set and repeat this process until all the tasks have been partitioned.

5.5.2 Performance w.r.t. Number Of Cores

In this experiment, we study the performance differences in terms of number of processors used by different approaches when scheduling given task sets. Three different test cases were generated and tested: 8 tasks, 16 tasks and 24 tasks. For

each test case, we randomly generated 50 task sets and set $DMP_{\Gamma} = 10\%$. The results are shown in Figure 5.1.

From Figure 5.1, we can see that the FF algorithm always utilizes more processors than all the other approaches. This is because it does not consider the harmonic relationships between tasks and therefore wastes processor resources. All four other approaches, by taking the task harmonic relationship into consideration, outperform FF significantly. It is interesting to see that the performance improvement increases following the order of H_m , H_v , H_{cdf} and H_{sum} .

The first two approaches, H_m and H_v have low computational overhead. However, they cannot accurately capture the harmonic relationships between tasks since they focus only on the expected values and variances of the tasks. The other two approaches, on the other hand, are more elaborative and have higher computational cost. However, they determine the harmonic relationship based on the entire distribution of a task and therefore can result in better mappings. As a result, we can see that, when task number is 16, the latter two approaches in average can save one core in scheduling the same task set. Moreover, with the increase of the task number, the flexibility to allocate tasks increases. Therefore, the performance improvement increases. For example, for 8 tasks, the improvement is about 0.5 core savings and for 24 tasks, the improvement becomes about 1.4 core savings.

5.5.3 Performance w.r.t. Schedulability

Next, we analyze the performance of different approaches in terms of schedulability. That is, for a given core number and task sets, how many task sets can be successfully scheduled. We used the same three test cases for the first experiment and set the core number to 5, 10 and 14 for 8 tasks, 16 tasks and 24 tasks, respectively. We

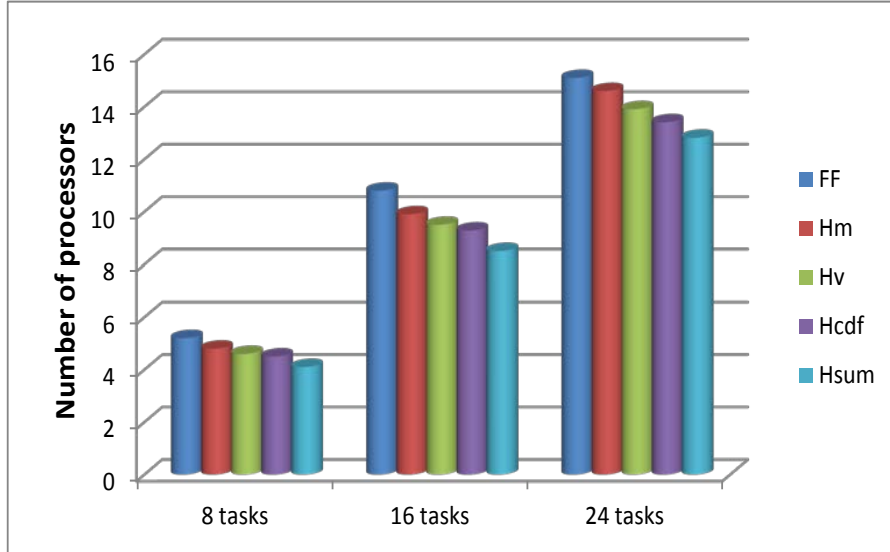


Figure 5.1: Processor usage versus task number with $DMP_T = 10\%$

show the results in Figure 5.2. Similar conclusions can be drawn from Figure 5.2 and H_{sum} has the highest scheduling rates among all five approaches. For 8 tasks, H_{sum} can improve upon FF by 30% and H_m by 16%. Also, with the increase of core numbers, the flexibility increases and therefore the performance is better. For example when there are 24 tasks in the task set, H_{sum} can improve upon FF by 45% and H_m by 28%.

5.5.4 Computational Costs

Finally, we want to compare the computational costs of approaches with different harmonic indexes. The results are shown in Figure 5.3. From the figure we can see that the more tasks we have the more time it takes for our harmonic-indexes-based approaches. For example, for 8 tasks, less than 4 seconds are needed for each approach to complete its computation while for 24 tasks, the fastest completion time is around 132 seconds (our approach with mean-based harmonic index, H_m).

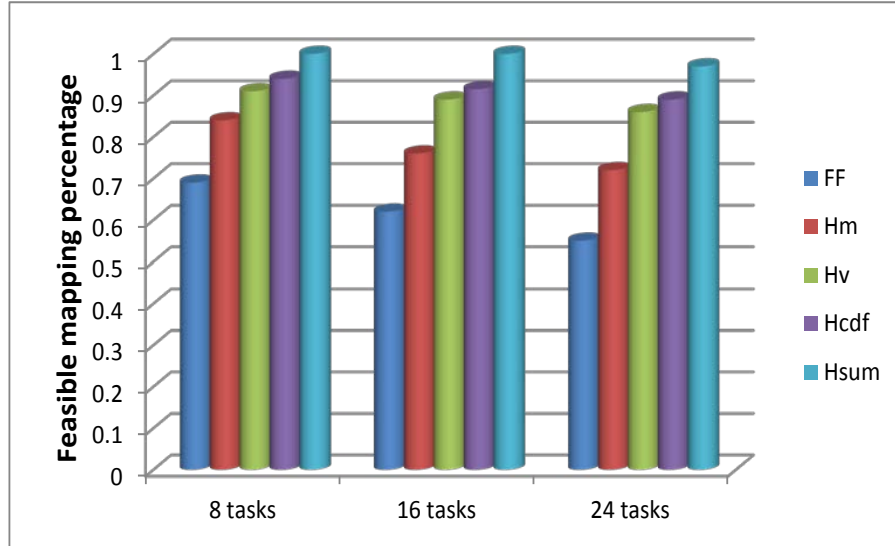


Figure 5.2: Feasible mapping percentages for different approaches

Moreover, H_{cdf} and H_{sum} have higher computational costs than H_m and H_v . Because H_{cdf} and H_{sum} need to calculate the difference between two distributions which takes more time compared with mean and variance calculations. Note that, FF can finish within 3 seconds for 24 tasks. However, our approach is an off-line scheme, the purpose is to minimize resource usage while satisfying the DMP constraints. The computational cost is not a major concern for this study.

5.6 Summary

With the increase of performance variations in modern computer systems, it is imperative to adopt a probabilistic approach rather than the traditional deterministic approach in the design and analysis of real-time systems. In this work, we develop a novel task partitioning algorithm for fixed-priority scheduling of real-time tasks with probabilistic execution times on a homogeneous multi-core platform with statistical guarantee. In our approach, we develop four novel metrics: *mean-based*,

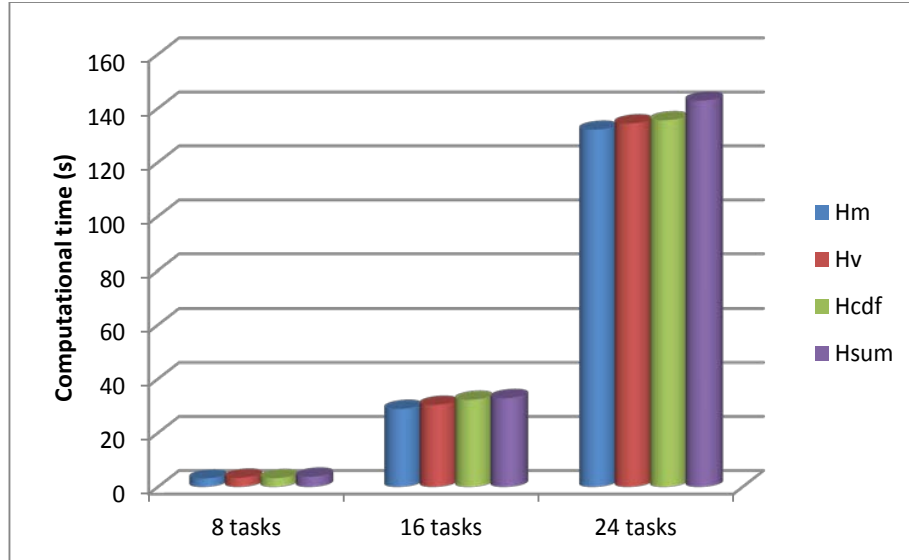


Figure 5.3: Computational costs of approaches with different harmonic indexes with $DMP_T = 10\%$

variance-based, cumulative-distribution-based and distribution-sum-based harmonic indexes to quantify the harmonicity among tasks, and based on these, better identify task set allocations and improve processor utilization. We conducted extensive simulation study and the results show that our algorithms can significantly outperform the existing approach.

Algorithm 7 Stochastic task partitioning algorithm.

Input:

- 1: 1) Task set: $\Gamma = \{\tau_1, \tau_2, \dots, \tau_N\}$;
- 2: 2) Task set deadline miss probability: DMP_Γ ;

Output:

- 3: Task partitions: $= \{subset_1, subset_2, \dots, subset_K\}$, K is the total number of processors.
 - 4: **while** $\Gamma \neq \emptyset$ **do**
 - 5: $SubSet = \emptyset$; //initialize the sub-set to empty
 - 6: $Probability = 0$; //initialize the probability of utilization distribution of a sub-set greater than threshold (0.7 in this case) to 0
 - 7: $\Gamma' = \{\Gamma'(\tau_1), \Gamma'(\tau_2), \dots, \Gamma'(\tau_L)\}$, // identify all the primary harmonic task sets, where L is the total number of primary harmonic task sets. If no two tasks such that their periods can satisfy $T_i|T_j$ ($i < j$), then $L = N$, otherwise $L < N$;
 - 8: **for** $i = 1$ to L // for each primary harmonic task set **do**
 - 9: $\Gamma'(\tau_i) = \{\tau'_1, \tau'_2, \dots, \tau'_N\}$ // sort the tasks with increasing order of \mathcal{H}_m , \mathcal{H}_v , \mathcal{H}_C or \mathcal{H}_S
 - 10: $subset_i = \emptyset$ // initialize a sub-set for $\Gamma'(\tau_i)$
 - 11: **for** $j = 1$ to N **do**
 - 12: $subset_i = subset_i + \tau'_j$
 - 13: **if** $DMP_{subset_i} > DMP_\Gamma$ **then**
 - 14: $subset_i = subset_i - \tau'_j$;
 - 15: **break**;
 - 16: **end if**
 - 17: **end for**
 - 18: **if** $Pr(\mathcal{U}_{subset_i} > 0.7) > Probability$ **then**
 - 19: $Probability = Pr(\mathcal{U}_{subset_i} > 0.7)$;
 - 20: $SubSet = subset_i$;
 - 21: **end if**
 - 22: **end for**
 - 23: $\Gamma = \Gamma - SubSet$;
 - 24: **end while**
-

CHAPTER 6

**ON THE HARMONIC FIXED-PRIORITY REAL-TIME TASKS
WITH EXPLICIT DEADLINES**

In the previous chapter, we proposed two task allocation approaches for statistical real-time tasks with deadline miss probability guarantee on a multi-core platform. In this chapter, we try to extend the work in Chapter 5 for tasks with explicit deadlines. Specifically, we adopt a partitioned approach for fixed-priority preemptive scheduling of real-time tasks with explicit deadlines on multi-core platforms with timing constraint guarantees. We develop two partitioning heuristics based on a novel metric that can quantify the degree of harmonicity between two tasks. Our extensive experimental results show that our approach can greatly improve the schedulability of real-time tasks when compared with existing works.

6.1 Related Work

When partition real-time tasks on multiple cores, one critical problem is how to partition real-time tasks such that processing cores can be most effectively utilized. High utilization usually implies opportunities for low implementation cost, low power/energy consumption, and high reliability of real-time systems. However, task partitioning is a well known NP-hard problem [38] and designers usually have to resort to different heuristics with low computational cost. One heuristic, for example, is simply to transform the problem into a bin-packing problem [35] and apply different bin-packing heuristics, such as first-fit (FF), worst-fit (WF) or best-fit (BF) to address this problem. Tasks are packed together without carefully considering their specifications, as long as they are schedulable on a single core. In the general case, Andersson et al [10] proved that the utilization bound for multi-core partitioned approach with fixed-priority scheduling is only 50% per core.

It is a well-known fact that harmonic tasks [70], i.e., tasks with periods being integer multiples of each other, can achieve high processor utilization on a single processor. A perfect harmonic periodic task set with implicit deadlines, i.e., tasks with deadlines equal to their periods, scheduled according to the rate monotonic scheduling (RMS) policy is schedulable as long as its total utilization is no more than 1. Therefore, many researchers have exploited this characteristic in developing more resource efficient real-time scheduling algorithms. For example, Kuo et al. [100] proposed a method to adjust loads on a single core processor by allocating harmonic tasks together. Han et al. [70] proposed a polynomial time method to determine the feasibility of a task set by verifying the feasibility of the corresponding harmonic task set transformed from the original task set. They proved that any task set that can pass the feasibility test by *Liu&Layland's* bound can also be validated by their proposed test. Recently, Bonifaci et al. [23] studied the feasibility for tasks with explicit deadlines on a uni-processor. They proved that when all the tasks have harmonic periods (i.e., any two tasks' periods pairwise divide each other), an exact polynomial-time algorithm for computing the response time of tasks can be found for both fixed priority scheduling and dynamic priority scheduling. Nasri et al. [127] presented a method to determine a set of harmonic periods among the possible values for tasks to simplify the worst case timing analysis and to improve the system utilization.

Besides extensive research on single core, there are also works that have been conducted to explore harmonic relationship on multi-core platforms. Liu et al. [112] studied the problem of scheduling harmonic tasks with suspensions on both uniprocessor and multiple processors. Fan et al. [46] proposed a partitioned scheduling algorithm to exploit harmonic relationship for fixed priority real-time tasks on multiprocessor platform. They extended their work to semi-partitioning algorithm that

can take advantage of harmonic relationships among tasks to improve the system schedulability [47, 49]. Wang et al. [166] further extended the task model with statistical execution times and proposed four metrics to quantify the harmonicity of task sets on multi-core platforms. All these works indicate that the system schedulability could be significantly improved if harmonic relationships among tasks can be exploited properly both for single core and multi-core platforms.

One great limitation of existing studies on harmonic real-time tasks is that they target solely on periodic tasks with implicit deadlines, and schedule (largely) according to RMS scheme. While previous work has clearly shown that taking harmonic relationship among tasks into consideration can be extremely beneficial in developing effective scheduling algorithms, such an approach is not applicable for many practical real-time applications [23] which can be better modeled as periodic real-time tasks with explicit deadlines.

To overcome this barrier, we extend the concept of “harmonic” from periodic tasks with implicit deadlines to the ones with explicit deadlines. We formally define what it means for tasks to be harmonic from the context of periodic tasks with explicit deadlines, scheduled according to deadline monotonic scheduling (DMS) policy. We show that, similar to a traditional harmonic task set, a *general* harmonic task set has a better schedulability than non-harmonic ones. Specifically, we formulate two theorems to demonstrate the high schedulability of a harmonic task set over a regular task set. To our best knowledge, this is the first research effort that defines the “harmonic task set” for periodic tasks with explicit deadlines.

We then take task harmonic relationship into consideration to tackle the problem of partitioned fixed-priority scheduling of real-time tasks on homogenous multi-core platforms based on DMS scheme. Since not all tasks are perfectly harmonic, we develop a novel metric to quantify the degree of harmonicity between two tasks.

Based on this metric, we then develop two partitioning algorithms that can take the harmonic relationship of tasks into consideration. Extensive simulations are conducted to validate our research, and results show that the proposed task partitioning approaches can significantly improve the schedulability of real-time tasks when compared with existing work.

The rest of the chapter is organized as follows. In Section 6.2 we introduce our system models and formally define our problem. We then use a motivation example to motivate our research problem. In section 6.3 we present an interesting finding for feasibility of tasks with explicit deadlines, and based on which we introduce our definition of harmonic tasks with explicit deadlines. We introduce the metric to quantify harmonic relationships between tasks in section 6.4. In Section 6.5 we present our task partitioning algorithms in detail. Section 6.6 presents the experimental results and finally we conclude in Section 6.7.

6.2 Preliminary

In this section we first introduce our system models such as real-time task models and processor models. Next, we formulate the problem formally. We then present a motivation example.

6.2.1 System Models And Problem Formulation

We consider a real-time system consisting of N independent periodic tasks, denoted as $\Gamma = \{\tau_1, \tau_2, \dots, \tau_N\}$, ordered by their priorities based on deadline monotonic scheduling (DMS) policy. Assume Γ is to be scheduled on a homogeneous multi-core platform, denoted as $P = \{p_1, p_2, \dots, p_M\}$, according to DMS. Each task $\tau_i \in \Gamma$ is characterized by a tuple (C_i, D_i, T_i) , representing the worst case execution time,

the relative deadline and the minimal inter-arrival time (period), respectively. A task is called an *implicit* deadline task when its deadline equals to its period, i.e., $D_i = T_i$, and called an *explicit* deadline task when its deadline is smaller than or equal to its period, $D_i \leq T_i$. Without loss of generality, we consider tasks with explicit deadlines in this work.

For each task $\tau_i = (C_i, D_i, T_i)$, we define its *intensity* (denoted as I_i) and *utilization* (denoted as U_i) as follows.

$$I_i = \frac{C_i}{D_i}, \quad (6.1)$$

$$U_i = \frac{C_i}{T_i}. \quad (6.2)$$

Accordingly, the *utilization of a task set* Γ , denoted as U_Γ , is formulated as below.

$$U_\Gamma = \sum_{i=1}^N U_i, \quad (6.3)$$

When task set Γ is scheduled on a multi-core system with K cores, we define the *system utilization* (denoted as U_s) as

$$U_s = \frac{U_\Gamma}{K}. \quad (6.4)$$

The problem of fixed-priority scheduling of periodic tasks with explicit deadlines on multi-core platforms can be formulated as follows:

Problem 6.2.1. *Given*

- *a task set consisting of N tasks, $\Gamma = \{\tau_1, \tau_2, \dots, \tau_N\}$,*
- *a multi-core platform with K homogeneous processing cores, $P = \{p_1, p_2, \dots, p_K\}$,*

partition the task set Γ on the multi-core platform and schedule the tasks on each core using DMS scheme such that all tasks can meet their deadlines and the number of cores used is minimized.

6.2.2 Motivations

A key to solve the problem stated above is to partition real-time tasks in a way that can best utilize the processors. Consider a task set with six tasks shown in Table 6.1.

Table 6.1: A task set with six tasks

τ_i	C_i	D_i	T_i	U_i
τ_1	1	2	4	0.25
τ_2	1	3	4	0.25
τ_3	1	4	6	0.17
τ_4	1	5	6	0.17
τ_5	7	12	29	0.24
τ_6	7	12	38	0.18

One simple approach to partition the tasks above is to transform it into a traditional bin packing problem. Then we can apply heuristics such as FF, WF or BF to partition the tasks to different cores. Let us use FF algorithm as an example. First, we order these tasks according to the decreasing order of their utilizations, i.e., $\{\tau_1, \tau_2, \tau_5, \tau_6, \tau_3, \tau_4\}$. Then we allocate the tasks one by one to the first core that can accommodate the task. For the above example, we have τ_1, τ_2, τ_3 and τ_4 allocated to core 1, τ_5 and τ_6 allocated alone to core 2 and core 3, respectively. As such, to schedule tasks in Table 6.1 based on FF approach, at least three processing cores are needed.

Since harmonic tasks can better utilize a processor, an intuitive approach is therefore to allocate tasks with same periods (or tasks with periods being integer multiples of each other) to the same core. Specifically, for the six tasks above, we assign task τ_1 and τ_2 together to one processor and task τ_3, τ_4 and τ_5 to another processor. Again, since task τ_6 cannot be assigned to either of the two processors,

we still have to utilize one more processor to schedule task τ_6 . It is not difficult to verify that, if we assign task τ_1, τ_3 and τ_5 to one core and τ_2, τ_4 and τ_6 to another core, we can feasibly schedule all six tasks in two cores.

Note that both approaches above have their limitations when considering tasks with explicit-deadlines. The first approach depends on the order of the tasks to make partitioning decisions while the latter only takes harmonicity of task periods into consideration when allocating tasks to different cores. Both approaches ignore the effects of deadline constraints for task partitioning. Also note that for a harmonic task set with implicit deadlines, any two tasks τ_i and τ_j can be combined into one single task τ_Z with $T_Z = \max\{T_i, T_j\}$ and $U_Z = U_i + U_j$, and thus the feasibility of the two tasks is equivalent to that of τ_Z [100]. With this transformation, the entire harmonic task set can be transformed into a task set with only one task. As long as the utilization of this task is no more than 1, the original task set is schedulable. However, when considering task set with explicit deadlines, such a transformation is no longer valid. Therefore, partitioning tasks with explicit deadlines based on their periods becomes ineffective.

We believe that, same as tasks with implicit deadlines, there must exist some *harmonic relationship* among tasks with explicit deadlines, and if this relationship is explored properly, we can greatly improve the processor utilization. The challenge is how to identify and quantify this relationship for periodic tasks with explicit deadlines. We discuss our approach for this problem in the sections that follow.

6.3 Harmonic Tasks With Explicit Deadlines

As discussed in Section 6.2.2, when tasks have explicit deadlines, the task set with harmonic periods does not necessarily have a better utilization of processor. The

question is then what type of task sets may have a better utilization? The following example can shed some light on this question.

Table 6.2: A task set with three tasks.

τ_i	C_i	D_i	T_i
τ_1	1	2	3
τ_2	1	3	4
τ_3	C_3	D_3	24

Consider the task set shown in Table 6.2. We assume that the execution times, the relative deadlines and the periods for task τ_1 and τ_2 are given, while for task τ_3 only its period is given. Note that when we change τ_3 's deadline D_3 , its largest schedulable execution time C_3 is also changing. The corresponding task intensity (see equation (6.2)) also varies. Table 6.3 lists different values of D_3 and corresponding C_3 and I_3 . Figure 6.1 also shows this relationship more intuitively. For example, when we set task τ_3 's deadline $D_3 = 8$, the corresponding largest execution time that can still make task τ_3 feasible is $C_3 = 3$. Therefore, the intensity $I_3 = \frac{C_3}{D_3} = \frac{3}{8} = 0.375$. As shown in both Table 6.3 and Figure 6.1, the intensity of task τ_3 does not vary with its deadline monotonically. It is interesting to note that task τ_3 's intensity achieves to its maximum when task τ_3 's deadline equals to 12 and 24, or the integer multiples of task periods from τ_1 and τ_2 . This seems to imply that when a lower priority task's deadline is integer multiples of all higher priority tasks' periods, the lower priority task may achieve its maximum intensity. It is not difficult to see that the higher the maximum intensity a task has, the more workload can be accommodated without compromising its deadline. The task therefore has a better schedulability. Based on this observation, we define the concept of *harmonic tasks* for periodic tasks with explicit deadlines as follows.

Table 6.3: Intensity changes for task τ_3 .

D_3	5	6	7	8	9	10	11	12	13	14
C_3	1	2	2	3	3	3	4	5	5	5
I_3	0.2	0.33	0.29	0.375	0.33	0.3	0.36	0.42	0.38	0.36
D_3	15	16	17	18	19	20	21	22	23	24
C_3	6	6	6	7	7	8	8	8	9	10
I_3	0.4	0.375	0.35	0.39	0.37	0.4	0.38	0.36	0.39	0.42

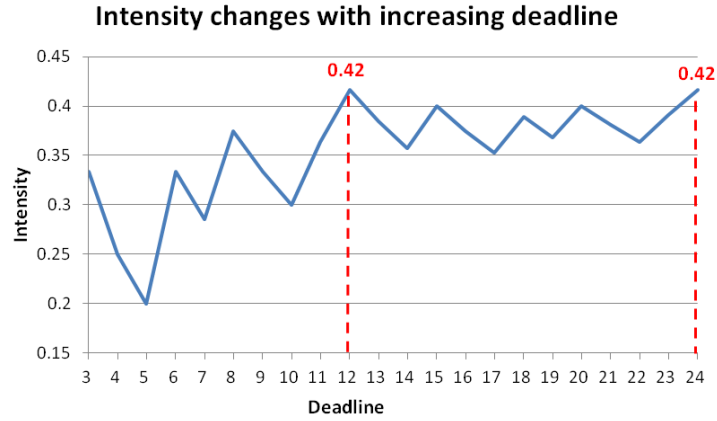


Figure 6.1: Intensity varies with different deadlines.

Definition 6.3.1. Let τ_i and τ_j be two tasks with explicit deadline with $D_i \leq D_j$.

Then τ_i and τ_j are harmonic if

- $T_i \leq D_j$ and $T_i \mid D_j$ (i.e., T_i divides D_j);
- $T_i > D_j$.

In Definition 6.3.1, if the deadline of the low priority task is the integer multiple of the period of the high priority task, then these two tasks are harmonic. This comes directly from the observation we introduce above. On the other hand, if the high priority task's period is larger than the deadline of the low priority task, we also define the two tasks being harmonic. This is because once the execution time of the

high priority task is given, the intensity of the low priority task varies monotonically with its deadline, exhibiting the same behavior when the period of the high priority task equals to the deadline of the low priority task. Accordingly, we can define the harmonic task set for tasks with explicit deadlines as follows.

Definition 6.3.2. *A task set is called a general harmonic task set, or simply harmonic task set if any two tasks in the task set are harmonic.*

From Definition 6.3.1 and Definition 6.3.2 we can see that the traditional harmonic tasks with implicit deadlines are just special cases of general harmonic tasks. While it is well-known that for harmonic tasks with implicit deadlines, the utilization bound is 1, this is not true any more for a general harmonic task set. To study the schedulability of a general harmonic task set, we have the following theorem.

Theorem 6.3.3. *Let task set $\Gamma = \{\tau_1, \tau_2, \dots, \tau_i, \dots, \tau_N\}$ be a harmonic task set with explicit deadlines. For $\tau_i \in \Gamma$, τ_i is schedulable if and only if the work demand of τ_i at the scheduling point $t = D_i$, i.e $W_i(D_i)$, is no more than D_i , where*

$$W_i(t) = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j. \quad (6.5)$$

Proof. The sufficiency of this statement is readily true. We only need to prove the necessity of the statement. Assume that there $\exists t \in [0, D_i)$ such that

$$W_i(t) = C_i + \sum_{\forall j < i} \left\lceil \frac{t}{T_j} \right\rceil \cdot C_j \leq t \quad (6.6)$$

Since $t < D_i$, t must be a scheduling point that is an integer multiple of a high priority task's period, e.g. T_j with $j < i$. Without loss of generality, let $t = kT_j$. In

the meantime, since τ_j and τ_i are harmonic, we have $D_i \mid T_j$. Let $D_i = mT_j$, and naturally we have $m > k$ and $D_i = \frac{m}{k}t$. Note that

$$W_i(D_i) = C_i + \sum_{\forall j < i} \left\lceil \frac{D_i}{T_j} \right\rceil \cdot C_j \quad (6.7)$$

$$= C_i + \sum_{\forall j < i} \frac{D_i}{T_j} \cdot C_j \quad (6.8)$$

$$= C_i + \frac{m}{k} \sum_{\forall j < i} \frac{t}{T_j} \cdot C_j \quad (6.9)$$

Since $\frac{m}{k} > 1$, $x \leq \lceil x \rceil$, and $W_i(t) \leq t$, we have

$$W_i(D_i) \leq \frac{m}{k}C_i + \frac{m}{k} \sum_{\forall j < i} \left\lceil \frac{t}{T_j} \right\rceil \cdot C_j \quad (6.10)$$

$$= \frac{m}{k}W_i(t) \quad (6.11)$$

$$\leq \frac{m}{k}t \quad (6.12)$$

$$= D_i. \quad (6.13)$$

Therefore τ_i must be schedulable. \square

From Theorem 6.3.3, we can see that, similar to traditional harmonic tasks, to check the schedulability of a task in a harmonic task set takes only linear time. More importantly, harmonic task sets defined by Definition 6.3.1 and 6.3.2 have better schedulability than non-harmonic ones. This conclusion is formulated in the following theorem and proved below.

Theorem 6.3.4. *Let $\Gamma = \{\tau_1, \tau_2, \dots, \tau_i, \dots, \tau_N\}$ and $\Gamma' = \{\tau'_1, \tau'_2, \dots, \tau'_i, \dots, \tau'_N\}$ be two schedulable task sets with equal utilization, i.e., $U_\Gamma = U_{\Gamma'}$. Assume that Γ is a perfect harmonic task set and Γ' is a regular task set. Let task $\tau_Z = (C_Z, D_Z, T_Z)$ be a task with priority lower than any task in Γ and Γ' . Then if $\{\Gamma' + \{\tau_Z\}\}$ is schedulable, then $\{\Gamma + \{\tau_Z\}\}$ must be schedulable.*

Proof. Since task τ_Z is feasible in task set $\{\Gamma' + \{\tau_Z\}\}$, there must exist a time instance $t \leq D_Z$ satisfy

$$C_Z + \sum_{i \leq N} \left\lceil \frac{t}{T'_i} \right\rceil \cdot C'_i \leq t. \quad (6.14)$$

Since $x \leq \lceil x \rceil$, we have

$$C_Z + \sum_{i \leq N} \frac{t}{T'_i} \cdot C'_i \leq t. \quad (6.15)$$

Then divided by t for both sides, we have

$$C_Z/t + \sum_{i \leq N} \frac{C'_i}{T'_i} \leq 1. \quad (6.16)$$

Since $t \leq D_Z$, we have

$$C_Z/D_Z + \sum_{i \leq N} \frac{C'_i}{T'_i} \leq 1. \quad (6.17)$$

Then we have,

$$C_Z + \sum_{i \leq N} \frac{D_Z}{T'_i} \cdot C'_i \leq D_Z. \quad (6.18)$$

or

$$C_Z + D_Z \sum_{i \leq N} U'_i = C_Z + D_Z U_{\Gamma'} \leq D_Z. \quad (6.19)$$

On the other hand, in task set $\{\Gamma + \{\tau_Z\}\}$, for τ_Z to be schedulable, we need

$$C_Z + \sum_{i \leq N} \left\lceil \frac{D_Z}{T_i} \right\rceil \cdot C_i \leq D_Z. \quad (6.20)$$

Since Γ is harmonic, we have

$$C_Z + \sum_{i \leq N} \frac{D_Z}{T_i} \cdot C_i = C_Z + D_Z U_{\Gamma} \leq D_Z. \quad (6.21)$$

Since $U_{\Gamma} = U_{\Gamma'}$, and based on Theorem 6.3.3 and equation (6.19), $\{\Gamma' + \{\tau_Z\}\}$ must be schedulable. \square

Theorem 6.3.4 indicates that for the same task τ_Z , if it is schedulable with a non-harmonic task set, it must be schedulable with a harmonic task set of the

same utilization. Also, for a harmonic task set and non-harmonic task set, if the corresponding tasks have the same utilizations and intensities, then if the non-harmonic task set is schedulable, the harmonic task set must be schedulable, as formally formulated in the following theorem.

Theorem 6.3.5. *Let $\Gamma = \{\tau_1, \tau_2, \dots, \tau_i, \dots, \tau_N\}$ and $\Gamma' = \{\tau'_1, \tau'_2, \dots, \tau'_i, \dots, \tau'_N\}$ be two task sets, and let $U_i = U'_i$ and $I_i = I'_i$ for any $\tau_i \in \Gamma$ and $\tau_i \in \Gamma'$. Assume Γ is perfect harmonic. Then if Γ' is schedulable, Γ must be schedulable.*

The proof for this theorem is very similar to that of Theorem 6.3.4 and thus omitted. It is not difficult to see that Theorem 6.3.5 can be applicable for traditional harmonic tasks with implicit deadlines. That is, if a task set is schedulable, then a harmonic task set with the same utilization must be schedulable. If tasks have explicit deadlines, however, we have to take the deadline constraints into consideration and require their intensities are equal.

Now let us revisit the motivation example. When we partition tasks based on the harmonic periods, three processors are needed: $\{\tau_1, \tau_2\}$, $\{\tau_3, \tau_4, \tau_5\}$ and $\{\tau_6\}$. The same task set can be scheduled using two processors: $\{\tau_1, \tau_3, \tau_5\}$ and $\{\tau_2, \tau_4, \tau_6\}$. If we pay close attention to the first subset $\{\tau_1, \tau_3, \tau_5\}$, we can see that τ_3 's deadline is an integer multiple of τ_1 's period, and task τ_5 's deadline is an integer multiples of periods for both τ_1 and τ_3 . Therefore this partition helps to reduce the number of processors. Similar observation can be made from the other subset. Note that τ_4 's deadline is very close to an integer multiple of τ_2 's period, and task τ_6 's deadline is integer multiples of periods for both τ_2 and τ_4 .

As the motivation example implies, if we take the harmonic relationship into consideration, we may significantly improve the processor utilization. At this time, we have formulated the harmonic relationship between tasks. However, not all tasks in a task set are perfectly harmonic. How can we quantify which tasks are more

harmonic than others? In what follows, we first introduce a metric to quantify the degree of harmonicity between two tasks. Based on this metric, we then propose two algorithms to guide our partition procedure on multi-core platforms.

6.4 Harmonic Index For Tasks With Explicit Deadlines

As shown in the previous section, a harmonic task set can have better resource usage than otherwise. However, not all task sets are strictly harmonic. Therefore, it is desirable to develop a metric to quantify how harmonic a task set is. In this section, we formally introduce the metric that we have developed to quantify the degree of harmonicity between two tasks.

We quantify the harmonicity of two tasks by measuring the “distance” of a task to the harmonic task. Before we introduce the metric in detail, we first introduce the following definitions.

Definition 6.4.1. *Given two tasks $\tau_i = (C_i, D_i, T_i)$ and $\tau_j = (C_j, D_j, T_j)$ with $D_i \leq T_i \leq D_j$, the harmonic sub-task of τ_j with respect to τ_i is task $\tau'_j = (C_j, D'_j, T_j)$, such that D'_j is the largest value with $D'_j \leq D_j$ and $D'_j \mid T_i$. On the other hand, the harmonic sub-task of τ_i with respect to τ_j is task $\tau'_i = (C_i, D_i, T'_i)$, such that T'_i is the largest value with $T'_i \leq T_i$ and $D_j \mid T'_i$.*

In other words, for the *low* priority task, its harmonic sub-task is the task with exactly the same execution time and period, but the largest possible *deadline* (not larger than the original one) that is perfectly harmonic with the high priority task. On the other hand, for the *high* priority task, its harmonic sub-task is the task with exactly the same execution time and deadline, but the largest possible period (not larger than the original one) that is perfectly harmonic with the low priority task. It is worthy of mentioning that we require the period of the low priority is no larger

than the deadline of the high priority task, i.e., $T_i \leq D_j$. We discuss the case when $T_i > D_j$ later. Moreover, the reason why we require the deadline (or period) of the harmonic sub-task is no larger than the original one is that, when replacing the original task with its harmonic sub-task and the task set is schedulable, the original task set must be schedulable. We formulate this conclusion in the following theorem.

Theorem 6.4.2. *Let $\Gamma = \{\tau_1, \tau_2, \dots, \tau_i, \dots, \tau_N\}$ and let τ'_i be a harmonic sub set with respect to any task $\tau_j \in \Gamma$. Then if task set $\{\tau_1, \tau_2, \dots, \tau'_i, \dots, \tau_N\}$ is feasible, Γ must be feasible.*

The proof for Theorem 6.4.2 can be easily obtained by recognizing that, for fixed-priority preemptive scheduling, increasing task period or deadline cannot compromise the schedulability of a task set.

Now we are ready to formally define a harmonic index to evaluate how a task is harmonic to the other.

Definition 6.4.3. *Given two tasks τ_i and τ_j with $D_i \leq T_i \leq D_j$, let τ'_j (τ'_i , resp) be the harmonic sub task of τ_j (τ_i , resp) with respect of τ_i (τ_j , resp). Then the harmonic index of τ_j (τ_i , resp) with respect of τ_i (τ_j , resp), denoted as $\mathcal{H}(\tau_j \rightarrow \tau_i)$ ($\mathcal{H}(\tau_i \rightarrow \tau_j)$, resp), is defined as*

$$\mathcal{H}(\tau_j \rightarrow \tau_i) = I'_j - I_j, \quad (6.22)$$

$$\mathcal{H}(\tau_i \rightarrow \tau_j) = U'_i - U_i. \quad (6.23)$$

where I'_j and I_j are intensities of τ'_j and τ_j , respectively, and U'_i and U_i are utilizations of τ'_i and τ_i , respectively.

The metrics of $\mathcal{H}(\tau_j \rightarrow \tau_i)$ and $\mathcal{H}(\tau_i \rightarrow \tau_j)$ define how close a task is to its corresponding harmonic sub-task in terms of intensity/utilization change. The larger the change is, the less harmonic the task is to the reference task. Therefore a high

harmonic index value indicates a low harmonic relationship. We also define the harmonic index to compare the harmonicity of different pairs of tasks as follows.

Definition 6.4.4. *Given two tasks τ_i and τ_j with $D_i \leq T_i \leq D_j$, the harmonic index of these two tasks, denoted as $\mathcal{H}(\tau_j, \tau_i)$ is defined as*

$$\mathcal{H}(\tau_i, \tau_j) = \min\{\mathcal{H}(\tau_j \rightarrow \tau_i), \mathcal{H}(\tau_i \rightarrow \tau_j)\}. \quad (6.24)$$

So far we have put our focus on the case when the high priority task's period is no larger than low priority task's deadline, i.e., $D_i \leq T_i \leq D_j$. If the high priority task's period is greater than low priority task's deadline, we consider the two tasks are perfectly harmonic. That is,

$$\mathcal{H}(\tau_j \rightarrow \tau_i) = \mathcal{H}(\tau_i \rightarrow \tau_j) = \mathcal{H}(\tau_i, \tau_j) = 0. \quad (6.25)$$

The rationale behind this definition is that, when a high priority task has a period longer than the deadline of the low priority task, the high priority task preempts the low priority task only one time, which is exactly the same when the high priority task has the period equal to the deadline of the low priority task.

We can extend the definition of the harmonic index for two tasks to the entire task set as follows.

Definition 6.4.5. *Given a task set $\Gamma = \{\tau_1, \tau_2, \dots, \tau_i, \dots, \tau_N\}$, the harmonic index of task τ_i in task set Γ , denoted as $\mathcal{H}_\Gamma(\tau_i)$, is defined as:*

$$\mathcal{H}_\Gamma(\tau_i) = \sum_{\tau_j \in \Gamma} \mathcal{H}(\tau_i, \tau_j) \quad (6.26)$$

$\mathcal{H}_\Gamma(\tau_i)$ describes the harmonic relationship of task τ_i in task set Γ by accumulating the harmonic index for all tasks in the task set. The smaller value means better harmonicity of the task to be allocated along with other tasks. In what follows, based on the metrics defined above, we proposed two algorithms to find tasks that are closer to harmonicity to assign to the same core.

6.5 Task Partitioning Algorithms For Tasks With Explicit Deadlines

With the harmonic indexes we have defined in the previous section, we are now ready to introduce our task partitioning algorithms. The goal of our task partitioning algorithms is to identify tasks that have high harmonicity and group them into one processor to better utilize resources. To this end, we propose two algorithms. The first algorithm, called *greedy intensity maximization algorithm* (GIM), allocates one task at a time to the core with the existing task set most harmonic to the task. The second algorithm, called *harmonic-aware clique maximization algorithm* (HCM), addresses the task partition problem from a higher perspective. It first identifies tasks that are harmonic or close to harmonic and then assigns them to a processing core together.

6.5.1 Greedy Intensity Maximization Algorithm

The greedy intensity maximization algorithm (GIM) is based on the harmonic index which we have proposed earlier. The details of the algorithm are shown in Algorithm 8. Given a task set and a multi-core platform, the algorithm first sorts the tasks in Γ in non-increasing order of their utilizations. Then it allocates each task to its best candidate processor starting from the top of the task queue. Specifically, for each iteration, GIM calculates the harmonic index of the task to the current task set at each core (Line 8-14), and then allocates the task to the processor that has the minimum harmonic index, i.e., $\mathcal{H}_{\Gamma_{p_j}}(\tau_i)$. Exact response time analysis is used to determine the schedulability of the task. If a task is unfeasible for all the avail-

able processors, the algorithm returns “**FAIL**” (Line 16-18). Otherwise, a feasible partition is generated (Line 20-23).

Algorithm 8 is a simple yet effective approach and the timing complexity is only $O(N^2M + Nmax(D))$, where N is the total number of tasks, M is the max number of cores, and $max(D)$ is the pseudo polynomial complexity for response time analysis. Since Algorithm 8 allocates one task at a time, the order of the tasks also plays a key role in the partitioning process. We sort the tasks in non-increasing order of their utilization because it is shown that such an order can usually achieve better results. The disadvantage of this approach is two-fold. First, since Algorithm 8 allocates one task at a time, it can only find a local optimal with limited choices, i.e., grouping with only tasks that have been assigned to a core. Second, the algorithm only minimizes the harmonic index of the task to tasks allocated to a core instead of maximizing the harmonicity of the whole task set allocated to the core. To this end, we propose another task partitioning algorithm based on the evaluation of the harmonicity of a group of tasks.

6.5.2 Harmonic-Aware Clique Maximization Algorithm

The second algorithm, harmonic-aware clique maximization algorithm (HCM), intends to identify tasks that have high harmonicity and allocate them to a core together. Different from GIM, HCM has more choices of tasks allocated to the same core and therefore can potentially achieve better performance.

To identify tasks with high harmonicity, one intuitive approach is to rank the harmonic indexes for all tasks based on each candidate task and then pick the ones with smaller harmonic indexes. However, different from the harmonic tasks with implicit deadlines, the general harmonic relationship is not transitive. From

Algorithm 8 Greedy intensity maximization algorithm.

Input:

- 1: 1) Task set: $\Gamma = \{\tau_1, \tau_2, \dots, \tau_N\}$;
- 2: 2) Multi-core platform: $P = \{p_1, p_2, \dots, p_M\}$;

Output:

- 3: Task partitions: $= \{\Gamma_{p_1}, \Gamma_{p_2}, \dots, \Gamma_{p_M}\}$. // Γ_{p_i} is the sub-task set on processor p_i
 - 4: sort the tasks in non-increasing order of their utilizations;
 - 5: **for** $i = 1$ to $|\Gamma|$ // for each task in Γ **do**
 - 6: flag = 0;
 - 7: threshold = ∞ ;
 - 8: temp = 0;
 - 9: **for** $j = 1$ to M // for each processor in P **do**
 - 10: **if** τ_i is feasible on p_j and $\mathcal{H}_{\Gamma_{p_j}}(\tau_i) < \text{threshold}$ **then**
 - 11: flag = 1;
 - 12: threshold = $\mathcal{H}_{\Gamma_{p_j}}(\tau_i)$;
 - 13: temp = j;
 - 14: **end if**
 - 15: **end for**
 - 16: **if** flag == 0 **then**
 - 17: break;
 - 18: **return** "FAIL";
 - 19: **else**
 - 20: $\Gamma_{p_{temp}} = \Gamma_{p_{temp}} + \tau_i$;
 - 21: **end if**
 - 22: **end for**
 - 23: **return** "SUCCESS" and task partitions;
-

Definition 6.3.1, it is not difficult to see that: if task A is perfectly harmonic to task B, and task B is perfectly harmonic to task C, it is not necessary that task A and task C be harmonic. Therefore, even though all tasks are selected in a way in which they are harmonic to the same task, the tasks are not necessarily harmonic to each other and eventually cause low resource utilization if they are allocated to the same core.

Our HCM is inspired by the classic maximum clique problem [22], which intends to find the largest fully connected subgraph in a graph. When different edges have different weights, the maximum clique problem can be transformed to find the sub-

graph with maximized/minimized total weights. In HCM, we let each task be a node in the graph and the harmonic index be the weight for the edge connecting two nodes. Then, the clique with the minimum total weight corresponds to the task set with the minimum harmonic index.

The maximum clique problem is a NP-hard problem in nature and many heuristics have been proposed, such as greedy algorithm, simulated annealing, neural network, etc [22] where the timing complexity is a serious concern. In HCM, we apply greedy-like heuristics to address this problem with a timing complexity of $O(N^4)$, where N is the total number of tasks. The details of the algorithm is shown in Algorithm 9.

HCM searches for a feasible clique in each iteration and returns the one with maximum utilization (Line 6-27). Specifically, for each iteration the algorithm constructs cliques with tasks that are harmonic or close to harmonic. Tasks are added to a clique until no further tasks can be added with all tasks being schedulable (Line 13-24). Then HCM sorts all candidate cliques in non-increasing order of their utilizations and picks the clique with the maximum utilization to allocate to a core. The tasks in the clique are then removed from task set Γ (Line 28-29). This process is repeated until task set Γ is empty.

6.6 Experimental Results

In this section, we use experiments to investigate the effectiveness of our proposed algorithms. Three sets of experiments are conducted. First, we compared the performance of different partitioning approaches in terms of acceptance ratios with respect to different system utilizations. Second, we compared acceptance ratios with different numbers of tasks on different processors. Third, we extended the execution time

of each task to a statistical model and evaluated the effectiveness of our proposed approach compared with existing works.

6.6.1 Experiment Setup

In our experiments, we randomly generated task sets based on UUniFast approach [21]. Specifically, we first fixed the total utilization, i.e., U_Γ , for the target task sets. Then for each task, its utilization U_i was generated to be uniformly distributed in the range of $[U_{min}, U_{max}]$. We chose U_{min} to be a very small number that is greater than 0 and U_{max} to be different values for different experiments. The period of a task was also randomly generated in the range of $[500, 1000]$. The deadline was generated according to the ratio $\frac{D_i}{T_i}$. Finally, based on parameters defined above, we randomly generated one task for each run and subtracted its utilization U_i from the total utilization U_Γ . Task generation terminated when $U_\Gamma = 0$.

Four different approaches were realized in our experiment, i.e., two task partitioning algorithms introduced above, one traditional bin packing approach, i.e., the first fit decreasing, and the harmonic approach that does not consider deadline constraints [70]. We denote our greedy intensity maximization algorithm as *GIM*, harmonic-aware clique maximization algorithm as *HCM*, first fit decreasing as *FF* and the one proposed in [70] as *DCT*. For *FF* approach, we sort the tasks according to their utilizations in non-increasing order and allocated a task to the first processor that can feasibly schedule it. If all tasks can be allocated successfully, the algorithm returns the schedulable task partition, and fails otherwise. For *DCT* approach, it searches for a harmonic task set based on the consideration of period harmonicity.

We use the acceptance ratio to evaluate the performance of each approach, where acceptance ratio is defined in Equation (6.27).

$$\text{acceptance ratio} = \frac{\text{schedulable task sets}}{\text{total number of task sets}} \quad (6.27)$$

6.6.2 Performance vs. System Utilizations

In this experiment, we studied the performance of different approaches (in terms of acceptance ratio) with respect to different system utilizations. We chose $U_{max} = 0.2$ for illustration purposes since in many practical systems, the majority of tasks were lightweigh. let $\frac{D_i}{T_i} = [0.2, 1]$ to cover a variety of tasks. Three different test cases with different numbers of processing cores were generated and tested: 4 processors, 8 processors and 12 processors. For each test case, we ran 1000 experiments and calculated the average acceptance ratio for different approaches. The results are shown in Figure 6.2.

From Figure 6.2, we can see that the two algorithms i.e., *GIM* and *HCM*, can always achieve higher acceptance ratios than *DCT* and *FF* and the performance improvement increases with the increase of core number. For 4-processor case, *HCM* in average can achieve around 12% and 8% improvement over *DCT* and *FF*, respectively. For 8-processor case, *HCM* in average can achieve around 15% and 12% improvement over *DCT* and *FF*, respectively. For 12-processor case, *HCM* has around 19% and 16% improvement over *DCT* and *FF*, respectively.

DCT is the worst approach among the four. The reason is that *DCT* only considers harmonicity between tasks in terms of periods. When tasks have implicit deadlines, *DCT* is an effective heuristic since periods are the sole concern when partitioning tasks. However, when tasks have explicit deadlines, we have to take deadlines into consideration. From the experimental results, we also found that *FF* tends to perform better than *DCT* and has a low timing complexity. In comparison,

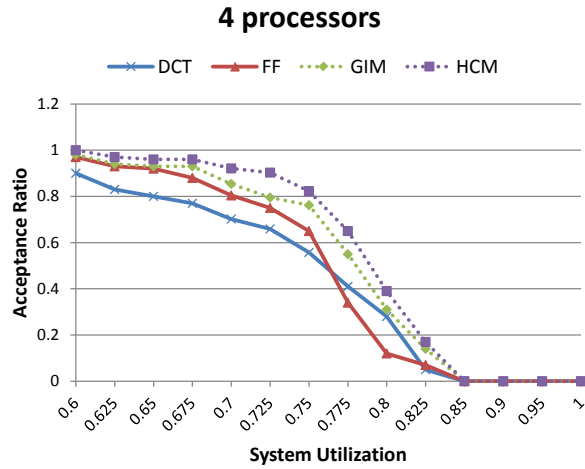
GIM also has a relatively low timing complexity as *FF*, but it can generate more feasible task partitions than *FF* does. *HCM* is better than all the other approaches for the reason that it searches the harmonic tasks from the entire task set and can best exploit the harmonic relationship among tasks.

We can also conclude from Figure 6.2 that the harmonic index we defined can accurately reflect the harmonicity between tasks with explicit deadlines. Finally, comparing Figure 6.2(a), Figure 6.2(b) and Figure 6.2(c), *HCM* can achieve better improvements over *DCT* and *FF* as both the utilization of task set and number of processors increase. This is because the more cores are available, the more opportunities there are for HCM to improve the performance.

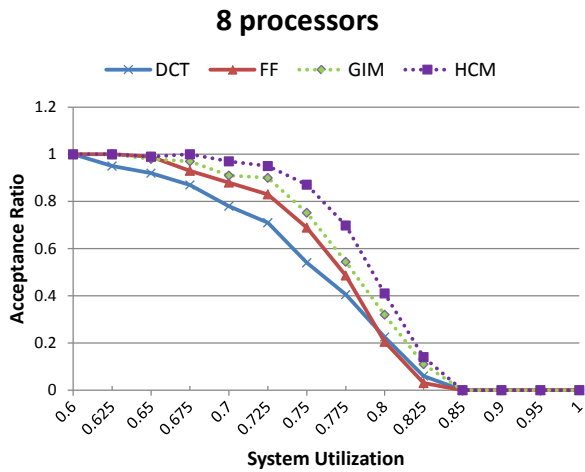
6.6.3 Performance vs. Number Of Tasks

In this experiment, we studied the performance of each approach with respect to different numbers of tasks. Specifically, we set $U_{max} = 0.2, 0.4, 0.6$ and 0.8 , respectively, and a task's deadline was randomly generated between its execution time and period. To generate a task, we first generated its period and utilization, where $T_i \in [500, 1000]$ and $U_i \in [0, U_{max}]$. Therefore, given the core number of a multi-core platform, the larger that U_{max} is, the smaller number of tasks can be scheduled on the platform. Next, the task's execution time can be calculated by its period and utilization. Finally, its deadline was generated between its execution time and period. Three test cases were performed on 4 processors, 8 processors and 12 processors. Each case was run for 1000 times and average acceptance ratio was calculated. The results are shown in Figure 6.3, 6.4 and 6.5.

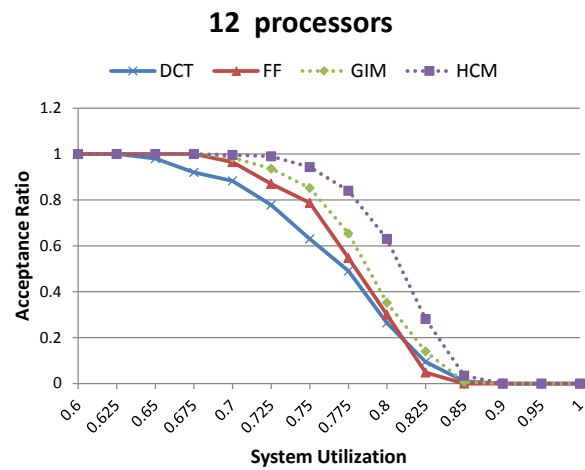
From the experimental results shown in Figure 6.3, 6.4 and 6.5, we can make the following observations. First, we can see that *HCM* algorithm always produces



(a) Acceptance ratio for each approach with 4 processors.



(b) Acceptance ratio for each approach with 8 processors.



(c) Acceptance ratio for each approach with 12 processors.

Figure 6.2: Performance vs. system utilizations

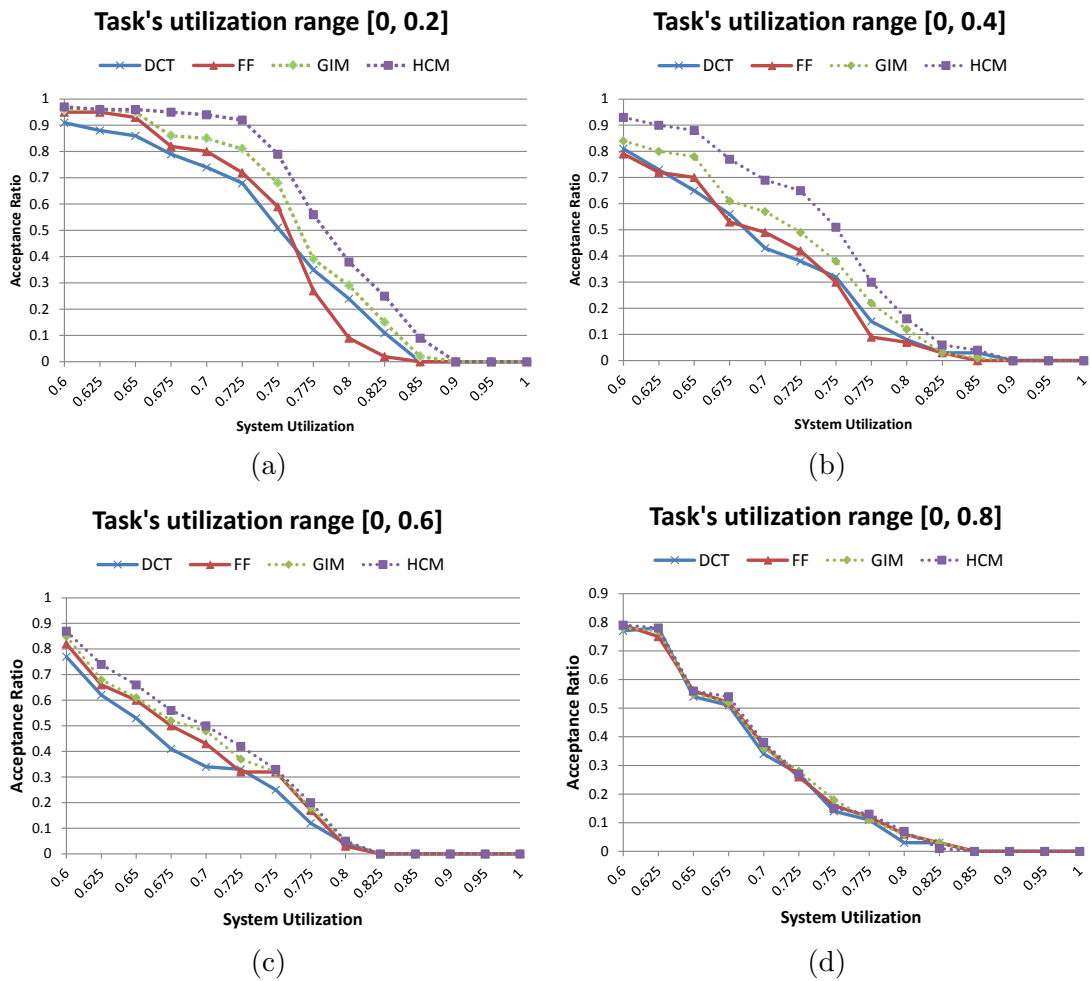


Figure 6.3: Performance vs. number of tasks on 4 processors

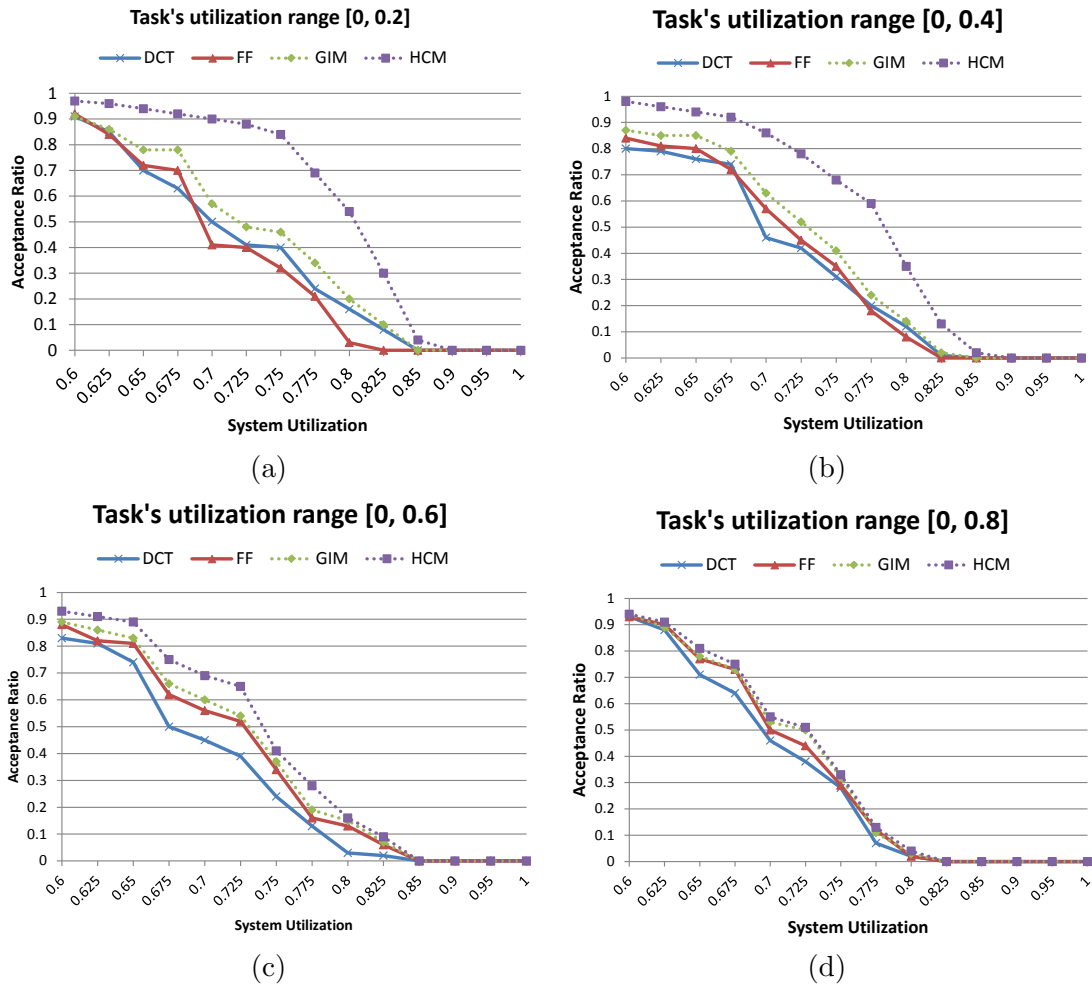


Figure 6.4: Performance vs. number of tasks on 8 processors

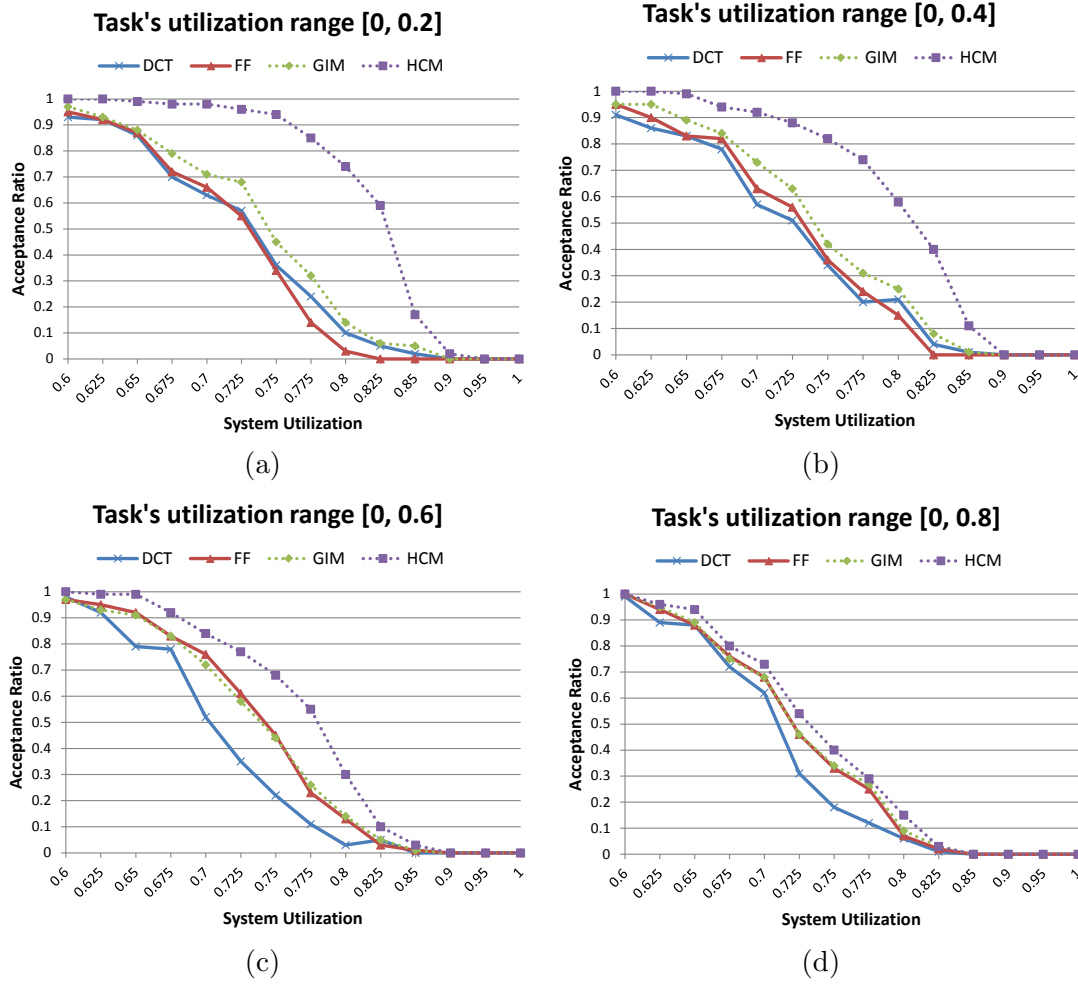


Figure 6.5: Performance vs. number of tasks on 12 processors

better results than all the others. When task utilization increases, the performance of each algorithm decreases. Due to the reason that when we increase task utilization range, for the same system utilization, there are less number of tasks available and therefore less flexibility for each algorithm to find a feasible solution. Also, as the task utilization increases, the improvement of *HCM* algorithm over other algorithms decreases because of the same reason. Second, when comparing Figure 6.3, 6.4 and 6.5, we can see that the improvement of *HCM* algorithm on 8 processors is better than that on 4 processors while the improvement on 12 processors is the best of all. The reason is that *HCM* can better utilize the available resources to allocate tasks than other algorithms. Third, we may notice that when the task utilization range is $[0, 0.2]$, the performance of all algorithms decreased except *HCM* algorithm compared to the experiment in the previous subsection. For example, let us compare Figure 6.2(c) and Figure 6.5(a). The performance of all other three algorithms has dropped significantly while the performance of *HCM* is improved. The reason is that for the previous experiment, we set deadline to period ratio as $[0.2, 1]$. However, in this experiment the deadline was randomly generated between execution time and period which means deadline to period ratio was further extended. *DCT* only considers period relationship, and *FF* makes decisions solely on utilization. Therefore, the larger the interval between deadline and period is, the less effective these two algorithms are. For *GIM*, it considers the harmonicity, but it assigns one task at a time. As a result, it potentially has a smaller exploration space than *HCM* which intends to pack tasks with close harmonic relationship into the same core.

6.6.4 Statistical Model Evaluation

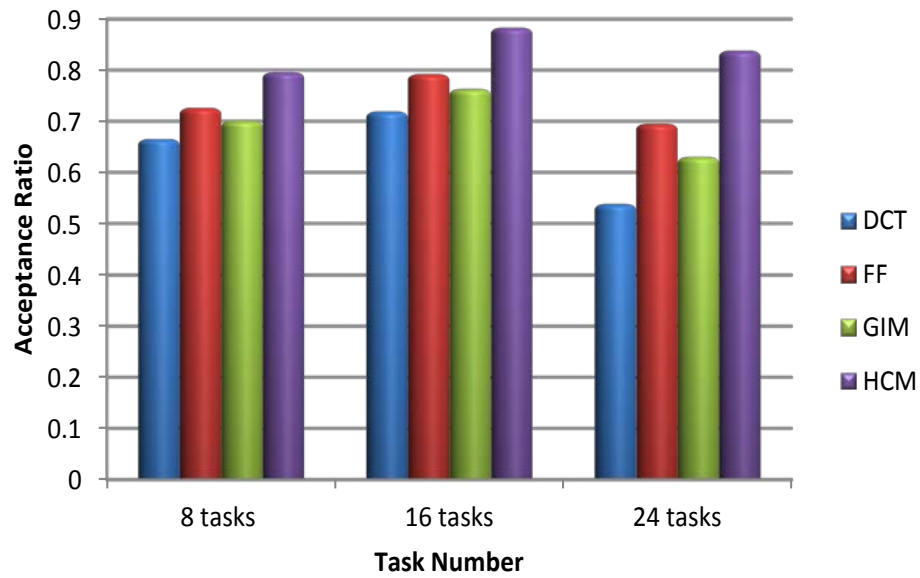
In this subsection, we want to extend our task model from deterministic to statistical. Specifically, we adopt a similar statistical task model as we did in the previous chapter. Each task has four different execution times along with a probability. A task's deadline and period however, remain deterministic. Then, we want to compare for a more general task model being defined in a statistical manner, the performance of our proposed approaches and existing works. Two test cases were generated with deadline miss probability of 5% and 10%. For each test case, a task set containing 8 tasks, 16 tasks and 24 tasks was evaluated. Specifically, a task set was generated in a way that each task's average utilization is in the range of $[0.2, 0.5]$. Then the four approaches, *DCT*, *FF*, *GIM*, and *HCM* were adopted to generate a feasible task allocations. Furthermore, we set a core usage constraint to see if the four approaches can generate a feasible solution using less or equal number of cores compared with the core usage constraint. We set 4 cores for 8 tasks, 8 cores for 16 tasks, and 11 cores for 24 tasks (The core usage constraints were chosen empirically).

The results are shown in Figure 6.6. From Figure 6.6(a), when deadline miss probability is set to 5%, we can see that *DCT*, which allocates tasks only according to their period relationships, has the worst acceptance ratio among all the four approaches. The reason is that *DCT* fails to take statistical information of tasks into consideration when partitioning tasks. Note that when task models are statistical, *FF* performs slightly better than *GIM*. The reason is that by allocating one task at a time, *GIM* heuristic can only exploit a small portion of the solution space. Moreover, the tasks are sorted according to their average utilizations and may benefit *FF* approach more than *GIM* approach. *HCM* has the best acceptance ratio among the four since it can exploit much more solution space than all the other three and therefore can make better use of harmonic index to partition tasks. Similarly, in

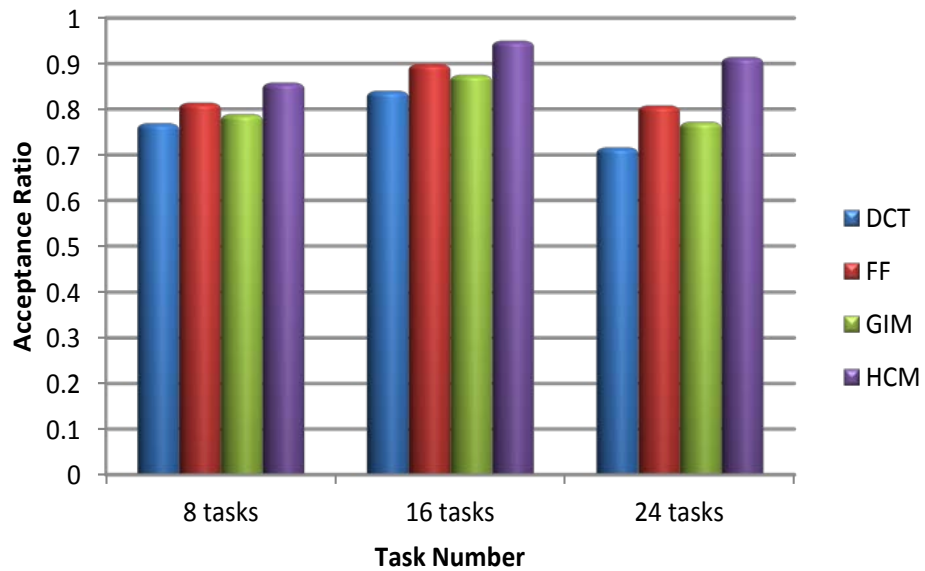
Figure 6.6(b) the acceptance ratios are better than the results shown in Figure 6.6(a) because the deadline miss probability is set to 10% and therefore, more task sets can be feasible for the same number of cores.

6.7 Summary

The continued evolution of IC technology and increasing complexity of real-time applications calls for innovative techniques to design real-time systems on multi-core platforms. One key problem to this end is how to partition tasks in a way that can most effectively utilize the resources. It is a well-known fact that a harmonic task set, i.e., task periods are integer multiples of each other, can better utilize a processor and achieve high system utilization. This feature has been exploited extensively in developing a variety of different real-time scheduling algorithms. However, a great limitation of these approaches is that the current definition of harmonic task set is limited only to real-time tasks with implicit deadlines. In this work, we extend the concept of “harmonic task set” to tasks with explicit deadlines and show that a *general* harmonic task set with explicit deadlines always has a better schedulability than a non-harmonic one. We employ this characteristic for task partitioning on multi-core, and extensive simulation results show that our algorithms can significantly outperform the existing approaches. As far as we know, this is the first research that defines the “harmonic task set” for periodic tasks with explicit deadlines. We believe that this research can greatly benefit many existing studies on harmonic task sets with implicit deadlines.



(a) Deadline miss probability is 5%.



(b) Deadline miss probability is 10%.

Figure 6.6: Deadline miss probability vs. number of tasks.

Algorithm 9 Harmonic-aware clique maximization algorithm.

Input:

- 1: 1) Task set: $\Gamma = \{\tau_1, \tau_2, \dots, \tau_N\}$;
- 2: 2) Multi-core platform: $P = \{p_1, p_2, \dots, p_M\}$;

Output:

- 3: Task partitions: $= \{\gamma_1, \gamma_2, \dots, \gamma_K\}$. // $K \leq M$ means a feasible solution is found
 - 4: **while** $\Gamma \neq \emptyset$ **do**
 - 5: Subset = \emptyset ;
 - 6: **for** $i = 1$ to $|\Gamma|$ **do**
 - 7: $subset_i = \{\tau_i\}$;
 - 8: $\hat{\Gamma} = \Gamma - \{\tau_i\}$;
 - 9: **while** 1 **do**
 - 10: flag = 0;
 - 11: threshold = ∞ ;
 - 12: temp = 0;
 - 13: **for** $j = 1$ to $|\hat{\Gamma}|$ **do**
 - 14: **if** τ_j is feasible in $subset_i$ and $\mathcal{H}_{subset_i}(\tau_j) < threshold$ **then**
 - 15: threshold = $\mathcal{H}_{subset_i}(\tau_j)$;
 - 16: temp = j ;
 - 17: flag = 1;
 - 18: **end if**
 - 19: **end for**
 - 20: **if** flag == 0 **then**
 - 21: break;
 - 22: **else**
 - 23: $subset_i = subset_i + \{\tau_{temp}\}$;
 - 24: $\hat{\Gamma} = \hat{\Gamma} - \{\tau_{temp}\}$;
 - 25: **end if**
 - 26: **end while**
 - 27: **end for**
 - 28: Subset = $max\{U_{subset_i}\}$; // U_{subset_i} is the utilization of $subset_i$
 - 29: $\Gamma = \Gamma - Subset$;
 - 30: **end while**
-

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

In this chapter, we summarize our contributions presented in this dissertation. We then discuss the possible directions for our future research work.

7.1 Summary

Driven by the need for massive performance power, multi-core platforms become mainstream as extensive research has been conducted from both academia and industry to exploit real-time scheduling algorithms on multi-core platforms. In this dissertation, we present our research work that has been done on real-time multi-core scheduling at system level.

First, we presented our real-time scheduling to partition DAG on a multi-core platform with the goal to minimize schedule length of the DAG with consideration of process variations. We introduced a virtualization framework that we can rely on to reconfigure the task allocations. Heuristics based on the concept of opportunity cost were introduced in this work. From our experimental studies, the proposed approach can achieve up to 30% and with an average of 15% of performance improvement (i.e. schedule length) by taking advantage of the heterogeneity of each individual platform.

Next, we extended our previous work in an attempt to address thermal issue as well. Based on the virtualization framework that we proposed earlier, we developed a fast temperature calculation equation given a periodic real-time schedule on a multi-core platform. Then we proposed three heuristics to re-map the speed schedule on each processor to minimize the overall peak temperature on chip. The proposed heuristics can achieve $14.09^{\circ}C$ temperature reduction in average and less than $5^{\circ}C$ of difference compared with *exhaustive search*. Overall, our proposed algorithm can

be finished within 1 second (more than 10^4 times faster compared to *exhaustive search*) which is the key to the success of optimization problems through *topology virtualization*.

Then we attacked the problem of statistical scheduling from the perspective of behavior model. We denoted each task’s execution time as a random variable instead of a deterministic value. We developed a novel task partitioning algorithm for fixed-priority scheduling of real-time tasks with probabilistic execution times on a homogeneous multi-core platform with statistical guarantee. Four novel metrics were proposed: *mean-based*, *variance-based*, *cumulative distribution-based* and *distribution sum-based* harmonic indexes to quantify the harmonic among tasks. Based on the four metrics, better task set allocations can be identified and processor utilization can be improved. We conducted an extensive simulation study, and the results show that our algorithms can significantly outperform the state-of-the-art approaches.

Finally, we extended our work to consider tasks with explicit deadlines and develop a novel metric that can explore the harmonicity between two tasks. Then we proposed two algorithms to partition real-time tasks with explicit deadlines on multi-core platforms that can better identify sub-task set partitions and save processor resource. This is the first research that defined the “harmonic task set” for periodic tasks with explicit deadlines. We believe that this research can greatly benefit existing research on harmonic task sets with implicit deadlines.

7.2 Future Work

Smaller feature sizes have enabled higher integration, faster switching and lower power consumption per transistor. According to Dennard scaling [39], as transistors

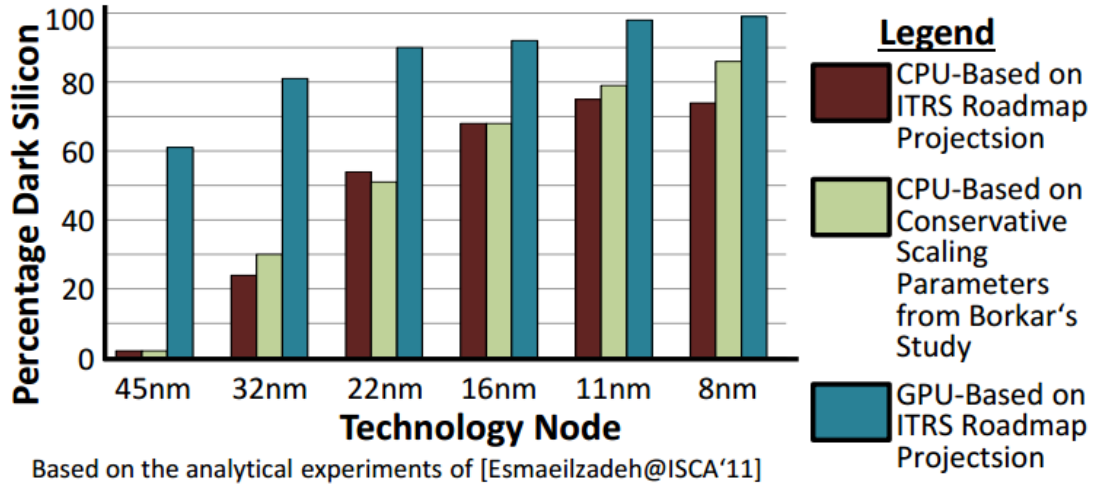


Figure 7.1: Dark silicon trends for different technology nodes [144].

get smaller, their power density remains approximately constant from one technology node to another. However, as we advance into deep sub-micron domain, leakage power dominates the total power consumption. As a result, threshold voltage cannot be scaled further without impacting performance, as reducing threshold voltage leads to exponential increase in leakage power. Therefore, the power density is trending upwards with technology scaling. It is projected that in the future, it will be only possible to power on a fraction of processors on a multi-core platform to satisfy the thermal design power (TDP) constraint which is the maximum amount of power that can be supplied to the chip to ensure that the chip will be operated within the safe range, i.e., below the thermal safe temperature, a term referred to as Dark Silicon Era [44, 155, 74]. Based on the information from ITRS and Intel, at the 8nm node, more than 50% of the chip area will be dark as shown in Figure 7.1.

Process variation affects conventional multi-core design in terms of operating frequency and leakage power dissipation. Therefore, the overall performance when executing multiple threads on a multi-core platform is limited by the thread running on the slowest core, and the “leaky” core (i.e., has higher leakage power consump-

tion) is much hotter than expected, leading to a higher peak temperature. Since in dark silicon era, only a portion of cores can be powered up at the same time, it can mitigate the problems that are brought by process variation. For example, we can choose which portion of cores to be powered on to meet all real-time constraints, and in the meanwhile, optimize design objectives such as power consumption and peak temperature.

Dark silicon is the next step that multi-core design is going to face in which only a fraction of cores are allowed to power on in order to maintain TDP constraint. However, it transforms variability from a concern to an opportunity that can be exploited. Specifically, in the presence of process variation, even homogeneous cores are heterogeneous in nature. This heterogeneity has been extensively studied in our previous research. Along with the redundancy brought by dark silicon, it is possible that more reliable systems can be guaranteed if we can properly extend our research in the Dark Silicon Era. However, one major challenge is that, with tens or hundreds of cores that are integrated on a single chip, it becomes difficult to effectively and efficiently explore the design space so that an optimal or near-optimal solution that is variable, reliable and thermally aware can be found.

In the future, we can extend our research on a variety of topics, such as reliability, temperature, and energy issues that are top concerns for real-time multi-core designs. As multi-core systems are going into dark silicon era, opportunities exist to explore more efficient and effective real-time scheduling approaches on multi-core platforms under uncertainty.

BIBLIOGRAPHY

- [1] Behind the birth of m^3 . *IHS iSuppli*, 2012.
- [2] A. 653. An avionics standard for safe, partitioned systems. In *Wind River Systems/IEEE Seminar*, 2008.
- [3] J. Aarestad, C. Lamech, J. Plusquellic, D. Acharyya, and K. Agarwal. Characterizing within-die and die-to-die delay variations introduced by process variations and soi history effect. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 534–539, Jun. 2011.
- [4] K. Agarwal and S. Nassif. Characterizing process variation in nanometer cmos. In *Design Automation Conference, 2007. DAC'07. 44th ACM/IEEE*, pages 396–399. IEEE, 2007.
- [5] B. Andersson. Global static-priority preemptive multiprocessor scheduling with utilization bound 38 In *Principles of Distributed Systems*, pages 73–88, 2008.
- [6] B. Andersson. Global static-priority preemptive multiprocessor scheduling with utilization bound 38%. In *Principles of Distributed Systems*, pages 73–88. Springer, 2008.
- [7] B. Andersson, S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In *Real-Time Systems Symposium, 2001. (RTSS 2001). Proceedings. 22nd IEEE*, pages 193–202, Dec 2001.
- [8] B. Andersson and A. Easwaran. Provably good multiprocessor scheduling with resource sharing. *Real-Time Systems*, 46(2):153–159, 2010.
- [9] B. Andersson and J. Jonsson. Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition. In *Real-Time Computing Systems and Applications, 2000. Proceedings. Seventh International Conference on*, pages 337–346. IEEE, 2000.
- [10] B. Andersson and J. Jonsson. The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%. In *2012 24th Euromicro Conference on Real-Time Systems*, pages 33–33. IEEE Computer Society, 2003.

- [11] B. Andersson and G. Raravi. Real-time scheduling with resource sharing on heterogeneous multiprocessors. *Real-Time Systems*, 50(2):270–314, 2014.
- [12] S. Anssi, S. Kuntz, S. Gérard, and F. Terrier. On the gap between schedulability tests and an automotive task model. *Journal of Systems Architecture*, 59(6):341–350, 2013.
- [13] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau. *Operating systems: Three easy pieces*. Popular Books Publishing, 2012.
- [14] A. Atlas and A. Bestavros. Statistical rate monotonic scheduling. In *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*, pages 123–132, Dec 1998.
- [15] N. C. Audsley, A. Burns, M. Richardson, and A. Wellings. *Deadline monotonic scheduling*. Citeseer, 1990.
- [16] P. Axer and R. Ernst. Stochastic response-time guarantee for non-preemptive, fixed-priority scheduling under errors. In *Design Automation Conference (DAC), 2013 50th ACM / EDAC / IEEE*, pages 1–7, May 2013.
- [17] M. Bao, A. Andrei, P. Eles, and Z. Peng. On-line thermal aware dynamic voltage scaling for energy optimization with frequency/temperature dependency consideration. In *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, pages 490–495, Jul. 2009.
- [18] 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. IEEE, 1990.
- [19] C. Batten. *Engineering: Complex digital asic design*. 2012.
- [20] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Syst.*, 30(1-2):129–154, May 2005.
- [21] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.
- [22] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo. The maximum clique problem. In *Handbook of combinatorial optimization*, pages 1–74. Springer, 1999.

- [23] V. Bonifaci, A. Marchetti-Spaccamela, N. Megow, and A. Wiese. Polynomial-time exact schedulability tests for harmonic real-time tasks. In *Real-Time Systems Symposium (RTSS), 2013 IEEE 34th*, pages 236–245. IEEE, 2013.
- [24] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter variations and impact on circuits and microarchitecture. In *Proceedings of the 40th annual Design Automation Conference*, pages 338–342, 2003.
- [25] V. Brocal, P. Balbastre, R. Ballester, and I. Ripoll. Task period selection to minimize hyperperiod. In *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on*, pages 1–4. IEEE, 2011.
- [26] J. M. Buchanan. Opportunity cost. In *The New Palgrave Dictionary of Economics Online (Second ed.)*, 2010.
- [27] J. V. Busquets-Mataix, J. J. Serrano, R. Ors, P. Gil, and A. Wellings. Using harmonic task-sets to increase the schedulable utilization of cache-based preemptive real-time systems. In *Real-Time Computing Systems and Applications, 1996. Proceedings., Third International Workshop on*, pages 195–202. IEEE, 1996.
- [28] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. *Handbook on scheduling algorithms, methods, and models*, pages 30–1, 2004.
- [29] K. Chakraborty and S. Roy. Rethinking threshold voltage assignment in 3d multicore designs. In *VLSI Design, 2010. VLSID '10. 23rd International Conference on*, pages 375–380, 2010.
- [30] T. Chantem, R. P. Dick, and X. S. Hu. Temperature-aware scheduling and assignment for hard real-time applications on mpsoCs. In *DATE*, pages 288–293, 2008.
- [31] T. Chantem, X. S. Hu, and R. Dick. Online work maximization under a peak temperature constraint. In *ISLPED*, pages 105–110, 2009.
- [32] V. Chaturvedi, H. Huang, and G. Quan. Leakage aware scheduling on maximal temperature minimization for periodic hard real-time systems. In *ICISS*, pages 1802–1809, 2010.

- [33] T. Chen and S. Naffziger. Comparison of adaptive body bias (abb) and adaptive supply voltage (asv) for improving delay and leakage under the presence of process variation. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 11(5):888–899, 2003.
- [34] G.-M. Chiu. The odd-even turn model for adaptive routing. *Parallel and Distributed Systems, IEEE Transactions on*, 11(7):729–738, 2000.
- [35] E. G. Coffman Jr, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. In *Approximation algorithms for NP-hard problems*, pages 46–93. PWS Publishing Co., 1996.
- [36] D. Dasari, B. Andersson, V. Nelis, S. M. Petters, A. Easwaran, and J. Lee. Response time analysis of cots-based multicores considering the contention on the shared memory bus. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*, pages 1068–1075. IEEE, 2011.
- [37] A. Datta, S. Bhunia, J. H. Choi, S. Mukhopadhyay, and K. Roy. Speed binning aware design methodology to improve profit under parameter variations. In *Design Automation, 2006. Asia and South Pacific Conference on*, pages 6–pp. IEEE, 2006.
- [38] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys (CSUR)*, 43(4):35, 2011.
- [39] 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. *Solid-State Circuits, IEEE Journal of*, 9(5):256–268, 1974.
- [40] S. K. Dhall and C. L. Liu. On a Real-Time Scheduling Problem. *Operations Research*, 26(1):127–140, 1978.
- [41] R. Dick, D. Rhodes, and W. Wolf. Tgff: task graphs for free. In *Hardware/Software Codesign, 1998. (CODES/CASHE ’98) Proceedings of the Sixth International Workshop on*, pages 97–101, Mar. 1998.
- [42] S. Edgar and A. Burns. Statistical analysis of wcet for scheduling. In *Real-Time Systems Symposium, 2001.(RTSS 2001). Proceedings. 22nd IEEE*, pages 215–224. IEEE, 2001.

- [43] S. Edgar and A. Burns. Statistical analysis of wcet for scheduling. In *Real-Time Systems Symposium, 2001. (RTSS 2001). Proceedings. 22nd IEEE*, pages 215–224, Dec 2001.
- [44] H. Esmailzadeh, E. Blem, R. St Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pages 365–376. IEEE, 2011.
- [45] M. Fan, Q. Han, G. Quan, and S. Ren. Multi-core partitioned scheduling for fixed-priority periodic real-time tasks with enhanced rbound. In *Quality Electronic Design (ISQED), 2014 15th International Symposium on*, pages 284–291. IEEE, 2014.
- [46] M. Fan and G. Quan. Harmonic-fit partitioned scheduling for fixed-priority real-time tasks on the multiprocessor platform. In *Embedded and Ubiquitous Computing (EUC), IFIP 9th International Conference on*, pages 27–32, Oct. 2011.
- [47] M. Fan and G. Quan. Harmonic semi-partitioned scheduling for fixed-priority real-time tasks on multi-core platform. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 503–508, March 2012.
- [48] M. Fan and G. Quan. Harmonic-aware multi-core scheduling for fixed-priority real-time systems. *Parallel and Distributed Systems, IEEE Transactions on*, 25(6):1476–1488, June 2014.
- [49] M. Fan and G. Quan. Harmonic-aware multi-core scheduling for fixed-priority real-time systems. volume 25, pages 1476–1488, June 2014.
- [50] N. Fisher, J.-J. Chen, S. Wang, and L. Thiele. Thermal-aware global real-time scheduling on multicore systems. *RTAS*, 0:131–140, 2009.
- [51] Y. Fu, N. Kottenstette, Y. Chen, C. Lu, X. D. Koutsoukos, and H. Wang. Feedback thermal control for real-time systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*, pages 111–120. IEEE, 2010.
- [52] M. R. Garey, R. L. Graham, and D. Johnson. Performance guarantees for scheduling algorithms. *Operations research*, 26(1):3–21, 1978.

- [53] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.
- [54] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.
- [55] M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 29. wh freeman, 2002.
- [56] S. Garg and D. Marculescu. System-level mitigation of wid leakage power variability using body-bias islands. In *Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis*, pages 273–278. ACM, 2008.
- [57] S. Garg and D. Marculescu. System-level leakage variability mitigation for mpsoe platforms using body-bias islands. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(12):2289–2301, 2012.
- [58] S. Garg and D. Marculescu. System-level leakage variability mitigation for mpsoe platforms using body-bias islands. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(12):2289–2301, 2012.
- [59] S. Garg, D. Marculescu, and S. X. Herbert. Process variation aware performance modeling and dynamic power management for multi-core systems. In *Proceedings of the International Conference on Computer-Aided Design*, pages 89–92, 2010.
- [60] D. Geer. Chip makers turn to multicore processors. *Computer*, 38(5):11–13, 2005.
- [61] C. Glass and L. Ni. The turn model for adaptive routing. In *Computer Architecture, 1992. Proceedings., The 19th Annual International Symposium on*, pages 278 –287, 1992.
- [62] A. Goel and P. Indyk. Stochastic load balancing and related problems. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 579–586. IEEE, 1999.
- [63] M. Gomaa, M. D. Powell, and T. N. Vijaykumar. Heat-and-run: leveraging smt and cmp to manage power density through the operating system. In

Proceedings of the 11th international conference on Architectural support for programming languages and operating systems, ASPLOS XI, pages 260–270, New York, NY, USA, 2004. ACM.

- [64] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287–326, 1979.
- [65] N. Guan, M. Stigge, W. Yi, and G. Yu. Fixed-Priority Multiprocessor Scheduling: Beyond Liu and Layland’s Utilization Bound. In *WiP Real-Time Systems Symposium (RTSS)*, Dec. 2010.
- [66] N. Guan, M. Stigge, W. Yi, and G. Yu. Fixed-Priority Multiprocessor Scheduling with Liu and Layland’s Utilization Bound. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Apr. 2010.
- [67] N. Guan, M. Stigge, W. Yi, and G. Yu. Parametric Utilization Bounds for Fixed-Priority Multiprocessor Scheduling. In *IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, May 2012.
- [68] R. Ha and J. W. S. Liu. Validating timing constraints in multiprocessor and distributed real-time systems. In *Distributed Computing Systems, 1994., Proceedings of the 14th International Conference on*, pages 162–171, Jun. 1994.
- [69] C.-C. Han, K.-J. Lin, and C.-J. Hou. Distance-constrained scheduling and its applications to real-time systems. *IEEE Trans. Comput.*, 45(7):814–826, July 1996.
- [70] C.-C. Han and H. ying Tyan. A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithms. In *Real-Time Systems Symposium, 1997. Proceedings., The 18th IEEE*, pages 36–45, Dec 1997.
- [71] Q. Han, M. Fan, and G. Quan. Energy minimization for fault tolerant real-time applications on multiprocessor platforms using checkpointing. In *Low Power Electronics and Design (ISLPED), 2013 IEEE International Symposium on*, pages 76–81, Sep. 2013.
- [72] V. Hanumaiah, R. Rao, S. Vrudhula, and K. S. Chatha. Throughput optimal task allocation under thermal constraints for multi-core processors. In *Proceedings of the 46th Annual Design Automation Conference, DAC '09*, pages 776–781, New York, NY, USA, 2009. ACM.

- [73] V. Hanumaiah, S. Vrudhula, and K. Chatha. Maximizing performance of thermally constrained multi-core processors by dynamic voltage and frequency control. pages 310–313, 2009.
- [74] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Toward dark silicon in servers. *IEEE Micro*, 31(EPFL-ARTICLE-168285):6–15, 2011.
- [75] D. S. Hochbaum et al. *Approximation algorithms for NP-hard problems*, volume 20. PWS publishing company Boston, 1997.
- [76] J. Howard, S. Dighe, and Y. Hoskote. A 48-core ia-32 message-passing processor with dvfs in 45nm cmos. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pages 108–109, Feb. 2010.
- [77] J. Hu and R. Marculescu. Dyad: smart routing for networks-on-chip. In *Proceedings of the 41st annual Design Automation Conference*, pages 260–263. ACM, 2004.
- [78] H. Huang, G. Quan, J. Fan, and M. Qiu. Throughput maximization for periodic real-time systems under the maximal temperature constraint. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 363 –368, Jun. 2011.
- [79] ITRS. *International Technology Roadmap for Semiconductors*. International SEMATECH, Austin, TX., <http://public.itrs.net/>.
- [80] ITRS. *International Technology Roadmap for Semiconductors (2008 Edition)*. International SEMATECH, Austin, TX., <http://public.itrs.net/>.
- [81] A. Jantsch, H. Tenhunen, et al. *Networks on chip*, volume 396. Springer, 2003.
- [82] JEDEC. *Failure mechanisms and models for semiconductor devices*. <http://www.jedec.org>, 2009.
- [83] C. Johnson and J. Welser. Future processors: flexible and modular. In *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 4–6. ACM, 2005.

- [84] D. Johnson, A. Demers, J. Ullman, M. Garey, and R. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974.
- [85] M. B. Jones. What really happened on mars, 1997.
- [86] H. Jung, P. Rong, and M. Pedram. Stochastic modeling of a thermally-managed multi-core system. In *Proceedings of the 45th annual Design Automation Conference*, pages 728–733. ACM, 2008.
- [87] S. Kato and N. Yamasaki. Portioned Static-Priority Scheduling on Multiprocessors. In *IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, Apr. 2008.
- [88] S. Kato and N. Yamasaki. Semi-Partitioned Fixed-Priority Scheduling on Multiprocessors. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Apr. 2009.
- [89] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell system technical journal*, 49(2):291–307, 1970.
- [90] C. H. Kim, K. Roy, S. Hsu, R. Krishnamurthy, and S. Borkar. A process variation compensating technique with an on-die leakage current sensor for nanometer scale dynamic circuits. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 14(6):646–649, 2006.
- [91] K. Kim, J. Diaz, L. Lo Bello, J. Lopez, C.-G. Lee, and S.-L. Min. An exact stochastic analysis of priority-driven periodic real-time systems and its approximations. *Computers, IEEE Transactions on*, 54(11):1460–1466, Nov 2005.
- [92] H. Kopetz. On the design of distributed time-triggered embedded systems. *Journal of Computing Science and Engineering*, 2(4):340–356, 2008.
- [93] H. Kopetz. *Real-time systems: design principles for distributed embedded applications*. Springer Science & Business Media, 2011.
- [94] K. Kuhn. Managing process variation in intel’s 45nm cmos technology. *Intel Technical Journal*, 12(2):93–110, Jun. 2008.

- [95] K. Kuhn. Cmos transistor scaling past 32nm and implications on variation. In *Advanced Semiconductor Manufacturing Conference (ASMC), 2010 IEEE/SEMI*, pages 241–246, Jul. 2010.
- [96] K. J. Kuhn. Reducing variation in advanced logic technologies: Approaches to process and design for manufacturability of nanoscale cmos. In *Electron Devices Meeting, 2007. IEDM 2007. IEEE International*, pages 471–474. IEEE, 2007.
- [97] K. J. Kuhn, M. D. Giles, D. Becher, P. Kolar, A. Kornfeld, R. Kotlyar, S. T. Ma, A. Maheshwari, and S. Mudanai. Process technology variation. *Electron Devices, IEEE Transactions on*, 58(8):2197–2208, 2011.
- [98] P. Kumar and L. Thiele. Thermally optimal stop-go scheduling of task graphs with real-time constraints. In *ASP-DAC*, pages 123–128, 2011.
- [99] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar. Body bias voltage computations for process and temperature compensation. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 16(3):249–262, 2008.
- [100] T.-W. Kuo and A. K. Mok. Load adjustment in adaptive real-time systems. In *Real-Time Systems Symposium, 1991. Proceedings., Twelfth*, pages 160–170. IEEE, 1991.
- [101] K. Lakshmanan, R. Rajkumar, and J. Lehoczky. Partitioned fixed-priority preemptive scheduling for multi-core processors. In *Euromicro Conference on Real-Time Systems (ECRTS)*, Jul. 2009.
- [102] C.-H. Lee and K. Shin. On-line dynamic voltage scaling for hard real-time systems using the edf algorithm. In *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*, pages 319 – 335, Dec. 2004.
- [103] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Real-Time Systems Symposium, 1990. Proceedings., 11th*, pages 201–209, Dec 1990.
- [104] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Real Time Systems Symposium, 1989., Proceedings.*, pages 166–171, Dec 1989.

- [105] T. Lei and S. Kumar. A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In *Digital System Design, 2003. Proceedings. Euromicro Symposium on*, pages 180–187, 2003.
- [106] D. W. Leinbaugh. Guaranteed response times in a hard-real-time environment. *Software Engineering, IEEE Transactions on*, (1):85–91, 1980.
- [107] J. Y.-T. Leung and M. Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Information processing letters*, 11(3):115–118, 1980.
- [108] H. Li, J. Sweeney, K. Ramamritham, R. Grupen, and P. Shenoy. Real-time support for mobile robotics. In *Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings. The 9th IEEE*, pages 10–18. IEEE, 2003.
- [109] K. Li, X. Tang, and K. Li. Energy-efficient stochastic task scheduling on heterogeneous computing systems. *Parallel and Distributed Systems, IEEE Transactions on*, 25(11):2867–2876, 2014.
- [110] K. Li, X. Tang, and B. Veeravalli. Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems. 2013.
- [111] W. Liao, L. He, and K. Lepak. Temperature and supply voltage aware performance and power modeling at microarchitecture level. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(7):1042 – 1053, 2005.
- [112] C. Liu, J. J. Chen, L. He, and Y. Gu. Analysis techniques for supporting harmonic real-time tasks with suspensions. In *Real-Time Systems (ECRTS), 2014 26th Euromicro Conference on*, pages 201–210, July 2014.
- [113] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM*, 20:46–61, Jan. 1973.
- [114] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, Jan. 1973.
- [115] S. Liu, J. Zhang, Q. Wu, and Q. Qiu. Thermal-aware job allocation and scheduling for three dimensional chip multiprocessor. In *International Symposium on Quality Electronic Design (ISQED), 2010*, pages 390–398, 2010.

- [116] J. Lopez, J. Daz, J. Entrialgo, and D. Garca. Stochastic analysis of real-time systems under preemptive priority-driven scheduling. *Real-Time Systems*, 40(2):180–207, 2008.
- [117] Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean. A statistical response-time analysis of real-time embedded systems. In *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd*, pages 351–362, Dec 2012.
- [118] C. Lung, Y. Ho, D. Kwai, and S. Chang. Thermal-aware online task allocation for 3d multi-core processor throughput optimization. In *Design, Automation, and Test in Europe (DATE)*, pages 1–6, Grenoble, France, 2011.
- [119] D. Maxim, O. Buffet, L. Santinelli, L. Cucu-Grosjean, and R. I. Davis. Optimal priority assignment algorithms for probabilistic real-time systems. In *RTNS*, pages 129–138. Citeseer, 2011.
- [120] D. Maxim and L. Cucu-Grosjean. Response time analysis for fixed-priority tasks with multiple probabilistic parameters. In *Real-Time Systems Symposium (RTSS), 2013 IEEE 34th*, pages 224–235, Dec 2013.
- [121] D. Maxim, M. Houston, L. Santinelli, G. Bernat, R. I. Davis, and L. Cucu-Grosjean. Re-sampling for statistical timing analysis of real-time systems. In *Proceedings of the 20th International Conference on Real-Time and Network Systems*, RTNS '12, pages 111–120, New York, NY, USA, 2012. ACM.
- [122] Z. Mengying, O. Alex, and X. C. Jason. Profit maximization through process variation aware high level synthesis with speed binning. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, pages 176–181, San Jose, CA, USA, 2013. EDA Consortium.
- [123] A. F. Mills and J. H. Anderson. A stochastic framework for multiprocessor soft real-time scheduling. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*, pages 311–320. IEEE, 2010.
- [124] A. K. Mok and D. Chen. A general model for real-time tasks. Technical report, Technical report, University of Texas at Austin, 1997.
- [125] A. K.-L. Mok and M. L. Dertouzos. *Multiprocessor scheduling in a hard real-time environment*. Domain Specific Systems Group [Massachusetts Institute of Technology], 1978.

- [126] M. Momtazpour, E. Sanaei, and M. Goudarzi. Power-yield optimization in mpsoC task scheduling under process variation. In *Quality Electronic Design (ISQED), 2010 11th International Symposium on*, pages 747–754, Mar. 2010.
- [127] M. Nasri, G. Fohler, and M. Kargahi. A framework to construct customized harmonic periods for real-time systems. In *Real-Time Systems (ECRTS), 2014 26th Euromicro Conference on*, pages 211–220. IEEE, 2014.
- [128] S. Nassif, K. Bernstein, D. Frank, A. Gattiker, W. Haensch, B. Ji, E. Nowak, D. Pearson, and N. Rohrer. High performance cmos variability in the 65nm regime and beyond. In *Electron Devices Meeting, 2007. IEDM 2007. IEEE International*, pages 569–571, 2007.
- [129] J. Oliver, R. Amirtharajah, V. Akella, and F. T. Chong. Credit-based dynamic reliability management using online wearout detection. In *Proceedings of the 5th conference on Computing frontiers*, pages 139–148, 2008.
- [130] S. Pandit and R. Shedge. Survey of real time scheduling algorithms. *OSR Journal of Computer Engineering*, 13(2):44–51, 2013.
- [131] J. Parkhurst, J. Darringer, and B. Grundmann. From single core to multi-core: preparing for a new exponential. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, pages 67–72. ACM, 2006.
- [132] G. Quan and V. Chaturvedi. Feasibility analysis for temperature-constraint hard real-time periodic tasks. *Industrial Informatics, IEEE Transactions on*, 6(3):329–339, 2010.
- [133] G. Quan and Y. Zhang. Leakage aware feasibility analysis for temperature-constrained hard real-time periodic tasks. *ECRTS*, pages 207–216, 2009.
- [134] P. Radojković, S. Girbal, A. Grasset, E. Quiñones, S. Yehia, and F. J. Cazorla. On the evaluation of the impact of shared resources in multithreaded cots processors in time-critical environments. *ACM Transactions on Architecture and Code Optimization (TACO)*, 8(4):34, 2012.
- [135] B. Raghunathan, Y. Turakhia, S. Garg, and D. Marculescu. Cherry-picking: exploiting process variations in dark-silicon homogeneous chip multiprocessors. In *Proceedings of the Conference on Design, Automation and Test in Europe, DATE '13*, pages 39–44, San Jose, CA, USA, 2013. EDA Consortium.

- [136] G. Raravi, V. Nélis, and B. Andersson. Real-time scheduling with resource sharing on uniform multiprocessors. In *Proceedings of the 20th International Conference on Real-Time and Network Systems*, pages 121–130. ACM, 2012.
- [137] I. Ripoll and R. Ballester-Ripoll. Period selection for minimal hyperperiod in periodic task systems. *Computers, IEEE Transactions on*, 62(9):1813–1822, 2013.
- [138] S. Saha, Y. Lu, and J. Deogun. Thermal-constrained energy-aware partitioning for heterogeneous multi-core multiprocessor real-time systems. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2012 IEEE 18th International Conference on*, pages 41–50, 2012.
- [139] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. Varius: A model of process variation and resulting timing errors for microarchitects. *Semiconductor Manufacturing, IEEE Transactions on*, 21(1):3–13, 2008.
- [140] J. Sartori, A. Pant, R. Kumar, and P. Gupta. Variation-aware speed binning of multi-core processors. In *Quality Electronic Design (ISQED), 2010 11th International Symposium on*, pages 307–314. IEEE, 2010.
- [141] J. Sartori, A. Pant, R. Kumar, and P. Gupta. Variation-aware speed binning of multi-core processors. In *Quality Electronic Design (ISQED), 2010 11th International Symposium on*, pages 307–314, Mar. 2010.
- [142] R. R. Schaller. Moore’s law: past, present and future. *Spectrum, IEEE*, 34(6):52–59, 1997.
- [143] L. Seiler and D. Carmean. Larrabee: A many-core x86 architecture for visual computing. *Micro, IEEE*, 29(1):10–21, Jan. 2009.
- [144] M. Shafique, S. Garg, J. Henkel, and D. Marculescu. The eda challenges in the dark silicon era. In *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*, pages 1–6. IEEE, 2014.
- [145] S. Sharifi, R. Ayoub, and T. Rosing. Tempomp: Integrated prediction and management of temperature in heterogeneous mpsocs. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 593–598, 2012.

- [146] C.-S. Shih, S. Gopalakrishnan, P. Ganti, M. Caccamo, and L. Sha. Scheduling real-time dwells using tasks with synthetic periods. In *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, pages 210–219. IEEE, 2003.
- [147] K. Shin and P. Ramanathan. Real-Time Computing: A New Discipline of Computer Science and Engineering. *Proc. IEEE*, 82(1):6–24, Jan. 1994.
- [148] P. Shivakumar, S. W. Keckler, C. R. Moore, and D. Burger. Exploiting microarchitectural redundancy for defect tolerance. In *Proceedings of the 21st International Conference on Computer Design, ICCD '03*, pages 481–. IEEE Computer Society, 2003.
- [149] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware computer systems: opportunities and challenges. *IEEE Micro*, 23(6):52–61, 2003.
- [150] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, pages 2–13, 2003.
- [151] E. Sperling. Turn down the heat ... please. *Electronic Design, Strategy, News*, page 1, Jul. 2006.
- [152] J. Stankovic. Misconceptions about real-time computing: a serious problem for next-generation systems. *Computer*, 21(10):10–19, 1988.
- [153] P. A. Stolk, F. P. Widdershoven, and D. Klaassen. Modeling statistical dopant fluctuations in mos transistors. *Electron Devices, IEEE Transactions on*, 45(9):1960–1971, 1998.
- [154] Y. Taur. Cmos design near the limit of scaling. *IBM Journal of Research and Development*, 46(2.3):213–222, Mar. 2002.
- [155] M. B. Taylor. Is dark silicon useful?: harnessing the four horsemen of the coming dark silicon apocalypse. In *Proceedings of the 49th Annual Design Automation Conference*, pages 1131–1136. ACM, 2012.
- [156] U. Tech. 2014 embedded market study. *Then, Now: What's Next?*, 2014.

- [157] K. Thulasiraman and M. N. Swamy. *Graphs: theory and algorithms*. John Wiley & Sons, 2011.
- [158] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J. W. S. Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Real-Time Technology and Applications Symposium, 1995. Proceedings*, pages 164–173, May 1995.
- [159] J. W. Tschanz, J. T. Kao, S. G. Narendra, R. Nair, D. A. Antoniadis, A. P. Chandrakasan, and V. De. Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage. *Solid-State Circuits, IEEE Journal of*, 37(11):1396–1402, 2002.
- [160] I. Ukhov, M. Bao, P. Eles, and Z. Peng. Steady-state dynamic temperature analysis and reliability optimization for embedded multiprocessor systems. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pages 197–204, 2012.
- [161] S. Vangal, J. Howard, and G. Ruhl. An 80-tile sub-100-w teraflops processor in 65-nm cmos. *Solid-State Circuits, IEEE Journal of*, 43(1):29–41, Jan. 2008.
- [162] F. Wang, Y. Chen, C. Nicopoulos, X. Wu, Y. Xie, and N. Vijaykrishnan. Variation-aware task and communication mapping for mpsoC architecture. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 30(2):295–307, Feb 2011.
- [163] F. Wang, Y. Chen, C. Nicopoulos, X. Wu, Y. Xie, and N. Vijaykrishnan. Variation-aware task and communication mapping for mpsoC architecture. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 30(2):295–307, Feb. 2011.
- [164] F. Wang, C. Nicopoulos, X. Wu, Y. Xie, and N. Vijaykrishnan. Variation-aware task allocation and scheduling for mpsoC. In *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, pages 598–603, Nov. 2007.
- [165] T. Wang, M. Fan, G. Quan, and S. Ren. Heterogeneity exploration for peak temperature reduction on multi-core platforms. In *Quality Electronic Design (ISQED), 2014 15th International Symposium on*, pages 107–114. IEEE, 2014.
- [166] T. Wang, L. Niu, S. Ren, and G. Quan. Multi-core fixed-priority scheduling of real-time tasks with statistical deadline guarantee. In *Proceedings of the*

2015 Design, Automation & Test in Europe Conference & Exhibition, pages 1335–1340. EDA Consortium, 2015.

- [167] T. Wang, G. Quan, S. Ren, and M. Qiu. Topology virtualization for throughput maximization on many-core platforms. In *Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on*, pages 408–415, 2012.
- [168] L. Wanner, C. Apte, R. Balani, P. Gupta, and M. Srivastava. Hardware variability-aware duty cycling for embedded sensors. volume 21, pages 1000–1012. IEEE, 2013.
- [169] S. Yaldiz, A. Demir, and S. Tasiran. Stochastic modeling and optimization for energy management in multicore systems: a video decoding case study. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(7):1264–1277, 2008.
- [170] C.-Y. Yang, J.-J. Chen, L. Thiele, and T.-W. Kuo. Energy-efficient real-time task scheduling with temperature-dependent leakage. In *DATE*, pages 9–14, 2010.
- [171] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *FOCS*, pages 374–382, 1995.
- [172] L.-T. Yeh and R. C. Chu. *Thermal Management of Microelectronic Equipment: Heat Transfer Theory, Analysis Methods, and Design Practices*. ASME Press, New York, NY, 2002.
- [173] L. Yuan, S. Leventhal, and G. Qu. Temperature-aware leakage minimization technique for real-time systems. In *ICCAD*, pages 761–764, 2006.
- [174] K. Yue, S. Ghalim, Z. Li, F. Lockom, S. Ren, L. Zhang, and X. Li. A greedy approach to tolerate defect cores for multimedia applications. In *Embedded Systems for Real-Time Multimedia (ESTIMedia), 2011 9th IEEE Symposium on*, pages 112–119, Oct. 2011.
- [175] K. Yue, F. Lockom, Z. Li, S. Ghalim, S. Ren, L. Zhang, and X. Li. Hungarian algorithm based virtualization to maintain application timing similarity for defect-tolerant noc. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pages 493–498, Feb. 2012.

- [176] L. Zhang, Y. Han, Q. Xu, and X. Li. Defect tolerance in homogeneous many-core processors using core-level redundancy with unified topology. In *Design, Automation and Test in Europe, 2008. DATE '08*, pages 891–896, Mar. 2008.
- [177] L. Zhang, Y. Han, Q. Xu, X. wei Li, and H. Li. On topology reconfiguration for defect-tolerant noc-based homogeneous manycore systems. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 17(9):1173–1186, 2009.
- [178] L. Zhang, Y. Han, Q. Xu, X. wei Li, and H. Li. On topology reconfiguration for defect-tolerant noc-based homogeneous manycore systems. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 17(9):1173–1186, Sep. 2009.
- [179] L. Zhang, Y. Yu, J. Dong, Y. Han, S. Ren, and X. Li. Performance-asymmetry-aware topology virtualization for defect-tolerant noc-based many-core processors. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 1566–1571, Mar. 2010.
- [180] S. Zhang and K. S. Chatha. Approximation algorithm for the temperature-aware scheduling problem. In *ICCAD*, pages 281–288, 2007.
- [181] S. Zhang and K. S. Chatha. Thermal aware task sequencing on embedded processors. In *DAC*, pages 585 – 590, 2010.
- [182] X. Zhou, Y. Xu, Y. Du, Y. Zhang, and J. Yang. Thermal management for 3d processors via task scheduling. In *ICPP*, pages 115–122, 2008.

VITA

TIANYI WANG

2006	B.S., Software Engineering Beihang University Beijing, China
2009	M.S., Software Engineering Beihang University Beijing, China
2015	Ph.D., Electrical Engineering Florida International University Florida, USA

PUBLICATIONS

Tianyi Wang, Qiushi Han, Gang Quan, (2015). *On the harmonic fixed-priority real-time tasks with explicit deadlines*, Real Time Systems Symposium, (Submitted).

Qiushi Han, Tianyi Wang, Gang Quan, (2015). *Enhanced Fault-Tolerant Fixed-Priority Scheduling of Hard Real-Time Tasks on Multi-Core Platforms*, Embedded and Real-Time Computing Systems and Applications (RTCSA), (Accepted).

Tianyi Wang, Linwei Niu, Shaolei Ren, Gang Quan, (2015). *Multi-core fixed-priority scheduling of real-time tasks with statistical deadline guarantee*, Design, Automation & Test in Europe (DATE), 1335–1340.

Tianyi Wang, Ming Fan, Gang Quan, Shangping Ren, (2014). *Heterogeneity exploration for peak temperature reduction on multi-core platforms*, Quality Electronic Design, International Symposium on (ISQED), 107–114.

Tianyi Wang, Gang Quan, Shangping Ren, Meikang Qiu, (2012). *Topology Virtualization for Throughput Maximization on Many-Core Platforms*, Parallel and Distributed Systems (ICPADS), IEEE International Conference on, 408–415.