3-21-2018

Memory-Aware Scheduling for Fixed Priority Hard Real-Time Computing Systems

Gustavo A. Chaparro-Baquero
Florida International University, gchap002@fiu.edu

DOI: 10.25148/etd.FIDC004092

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

MEMORY-AWARE SCHEDULING FOR FIXED PRIORITY HARD REAL-TIME COMPUTING SYSTEMS

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

ELECTRICAL ENGINEERING

by

Gustavo A. Chaparro-Baquero

2018

To: Dean John L. Volakis
College of Engineering and Computing

This dissertation, written by Gustavo A. Chaparro-Baquero, and entitled Memory-Aware Scheduling for Fixed Priority Hard Real-Time Computing Systems, 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.

Jean Andrian

Nezih Pala

Deng Pan

Wujie Wen

Gang Quan, Major Professor

Date of Defense: March 21, 2018

The dissertation of Gustavo A. Chaparro-Baquero is approved.

Dean John L. Volakis
College of Engineering and Computing

Andrés G. Gil
Vice President for Research and Economic Development
and Dean of the University Graduate School

Florida International University, 2018

DEDICATION

I would like to dedicate this Doctoral dissertation to my beloved wife, Viky Arnedo, my dearest mother and aunt, Ana Lucía Baquero and Shirley Sanchez, and all my family. Without their love, understanding, support, and encouragement, the completion of this endeavor would never have been possible.

ACKNOWLEDGMENTS

First, I want to express my heartfelt appreciation to my major advisor, Dr. Gang Quan, for his constant guidance and encouragement during the last six years of my doctoral study. I also want to express my gratitude to my Ph.D. committee members, Dr. Jean Andrian, Dr. Nezih Pala, Dr. Deng Pan, and Dr. Wujie Wen, for their insightful feedback, comments and suggestions in improving the quality of this dissertation. I am proud to have such wonderful and knowledgeable people serving on my dissertation committee. In addition, I want to thank Dr. Gustavo Roig, Dr. Alexander Perez-Pons, Dr. Kang Yen, and Dr. Amaury Caballero for their unconditional and always kind advice.

I am thankful to the staff of the ECE department at FIU, specially to Mrs. Pat Brammer, Mr. Oscar Silveira, Mrs. Layla El-Hilu, Mrs. Mais Kayyali, and Mrs, Xiang Li for their great commitment to student services.

Next, I would like to thank my lab mates and friends, Dr. Soamar Homsi, Dr. Shi Sha, Dr. Ming Fan, Dr. Shuo Liu, Dr. Tianyi Wang, Dr. Qiushi Han, and Dr. Vivek Chaturvedi, for creating a wonderfully collaborative and friendly work environment.

Last, but not least, my deepest gratitude goes to all my family, sisters, cousins, aunts, uncles, in-laws, nieces, nephews, and friends, for their constant love and support during this journey. I am very grateful to my beloved wife, Mrs. Viky Arnedo, for accompanying and encouraging me through all these years. I want to give my life-long gratitude to my dearest mother and aunt, Mrs. Ana Lucía Baquero and Mrs. Shirley Sanchez, for all the love and affection they have showered upon me. I want to express also my gratitude to my friend Cavally for all his support.

ABSTRACT OF THE DISSERTATION

MEMORY-AWARE SCHEDULING FOR FIXED PRIORITY HARD REAL-TIME COMPUTING SYSTEMS

by

Gustavo A. Chaparro-Baquero

Florida International University, 2018

Miami, Florida

Professor Gang Quan, Major Professor

As a major component of a computing system, memory has been a key performance and power consumption bottleneck in computer system design. While processor speeds have been kept rising dramatically, the overall computing performance improvement of the entire system is limited by how fast the memory can feed instructions/data to processing units (i.e. so-called memory wall problem). The increasing transistor density and surging access demands from a rapidly growing number of processing cores also significantly elevate the power consumption of memory systems. In addition, the interference of memory accesses from different applications and processing cores significantly degrades the computation predictability, which is essential to ensure timing specifications in real-time system design. The recent IC technologies (such as 3D-IC technology) and emerging data-intensive real-time applications (such as Virtual Reality/Augmented Reality, Artificial Intelligence, Internet of Things) further amplify these challenges. We believe that it is not simply desirable but necessary to adopt a joint CPU/Memory resource management framework to deal with these grave challenges.

In this dissertation, we focus on studying how to schedule fixed-priority hard real-time tasks with memory impacts taken into considerations. We target on the fixed-priority real-time scheduling scheme since this is one of the most commonly used strategies for practical real-time applications. Specifically, we first develop an approach that takes into con-

sideration not only the execution time variations with cache allocations but also the task period relationship, showing a significant improvement in the feasibility of the system. We further study the problem of how to guarantee timing constraints for hard real-time systems under CPU and memory thermal constraints. We first study the problem under an architecture model with a single core and its main memory individually packaged. We develop a thermal model that can capture the thermal interaction between the processor and memory, and incorporate the periodic resource sever model into our scheduling framework to guarantee both the timing and thermal constraints. We further extend our research to the multi-core architectures with processing cores and memory devices integrated into a single 3D platform. To our best knowledge, this is the first research that can guarantee hard deadline constraints for real-time tasks under temperature constraints for both processing cores and memory devices. Extensive simulation results demonstrate that our proposed scheduling can improve significantly the feasibility of hard real-time systems under thermal constraints.

TABLE OF CONTENTS

CHAPTER PAGE

LIST OF TABLES

TABLE PAGE

LIST OF FIGURES

FIGURE PAGE

CHAPTER 1

INTRODUCTION

Computing systems are everywhere. Millions of computing systems are built every year destined to cover multiple needs in general purpose computing applications (e,g. personal computers, workstations, and servers) and specialized applications, known as embedded systems computing (e.g. portable devices, camcorders, and home appliances). Many computing systems, especially many embedded systems, require the execution of real-time processes, with the correctness depending not only on the logical correctness of the computational result, but also on the time such a result is produced. These kinds of systems are known as real-time systems. Real-time systems cover a wide spectrum of applications, from smart devices and systems (e.g. surveillance cameras, home automation systems, smart TVs, in-vehicle infotainment systems) to numerous more sophisticated real-time systems used to monitor and control physical systems and processes in many domains (e.g. manned and unmanned vehicles, critical infrastructures, process control systems in industrial plants, smart medical instruments, etc.) [14]. Thus, real-time systems are ubiquitous and their correct design is critical to almost every aspect of our daily life.

Memory system, as a major component in computing systems, has increasingly become a major barrier in real-time system design. Due to the rapid evolution of processors, and with the increasing adoption of multi-core platforms for multiple computing applications, more and more data travel back and forth between the processor and the memory. Therefore, the bandwidth of the memory (i.e. the speed of the memory) becomes one of the major constraints impacting the system performance [15].

Moreover, higher memory capacity and bandwidth requirements have increased the use of cache memories on the CPU, which increases the memory access latency variance and thus the unpredictability during the execution time for real-time tasks. Furthermore, a

larger amount of memory has also significantly increased power consumption and operating temperature. Today, how to effectively deal with these technical challenges presented by the memory system has been vital in the design of new generations of real-time computing systems [16–18].

Our research presented in this dissertation focuses on developing memory-aware resource allocation strategies for time-critical real-time systems. In this chapter, we first introduce the basics of real-time systems and real-time allocation strategies. Then, we discuss the challenges presented by memory systems in design of real-time systems. Next, we define our research problem and briefly summarize our contributions. Finally, we present the structure of this dissertation.

1.1 Real-time systems and real-time scheduling

In real-time computing domain, the correctness of a system depends not only on the logical result of the computation, but also on the time such a computation is produced [14]. A reaction that occurs too late could be useless or even dangerous. For example, many smart devices and systems (surveillance cameras, home automation systems, smart TVs, in vehicle infotainment systems) demand the capability of performing real-time computations. The correct design of real-time systems is becoming more and more critical to our society because there is an increasing number of complex computing systems requiring accurate and on-time computations. As an illustration, systems such as self-driving automotive controls and aircraft navigation systems require the execution of certain tasks in a timely manner, otherwise the consequences can be loss of human lives.

Real-time systems are ubiquitous and affect almost every aspect of our daily life. Real-time system designs cover a wide spectrum of applications, from relatively simple ones to extremely complex ones. In many cases, the real-time computer running the application is embedded into the system to be controlled. As we can see in Fig. 1.1, there

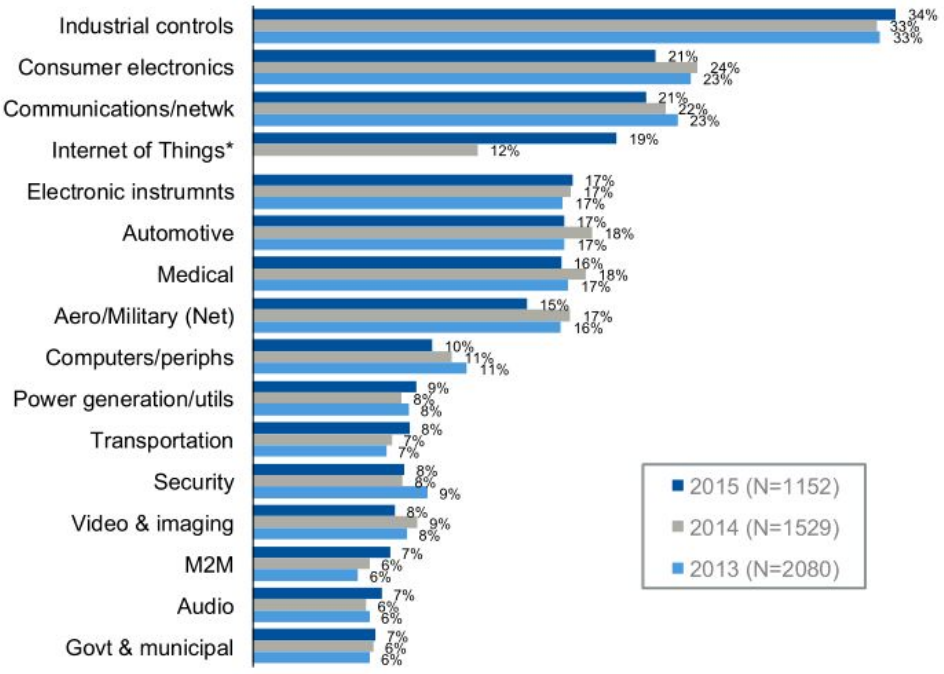


Figure 1.1: Embedded systems development industries in 2015 [3]

is a wide variety of industries where embedded systems are being developed. For example, as shown in Fig. 1.1, industrial controls is the industry implementing the larger number of embedded systems with 34% if its projects involving embedded systems in 2015. Consumer electronics, communications and internet-of-things (IoT) are the following industries with 21%, 21% and 19% of projects involving embedded systems in 2015, respectively. Based on a Zion Market Research report, the global embedded systems' market was valued at 159.00 billion USD in 2015, and is expected to generate a revenue of 225.34 billion USD by the end of 2021, growing at a Compound Annual Growth Rate (CAGR) of slightly above 6% between 2016 and 2021 [19]. In the meantime, as shown in Fig. 1.2, 62% of embedded systems on the market for 2015 were developed with real-time capabilities, highlighting the significant economic impacts of real-time systems.

A real-time system is still a computing system, and shares many common characteristics with other computing systems. However, what makes a real-time system unique is that

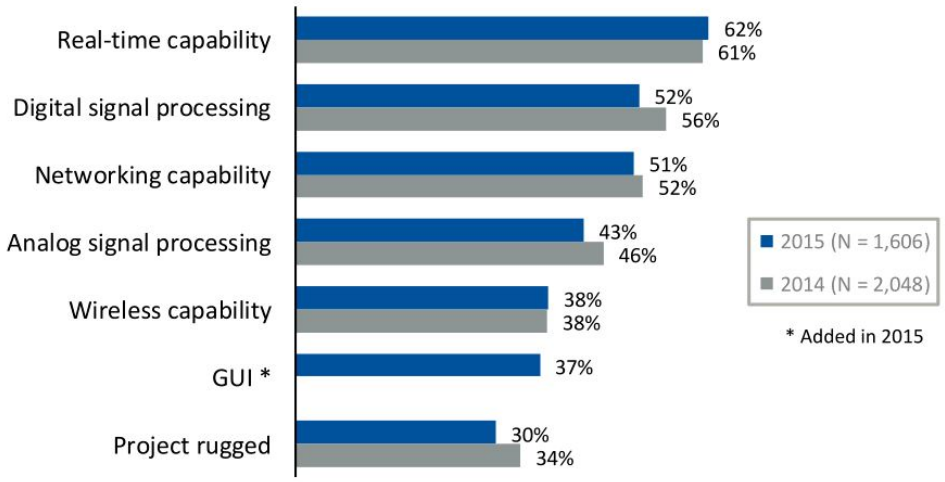


Figure 1.2: Current embedded systems capabilities in 2015 [3]

it is expected to guarantee a correct response within specified time constraints. In other words, processes executed on a real-time system are expected to be completed before their predetermined deadlines. Even for a logically correct result, if it is a "late result," it might be as negative as the incorrect output from the computation, leading to catastrophic consequences for certain applications. For instance, in a production line control system, various machines have to receive their orders at the right time to ensure smooth operation of a plant and to fulfill customer orders on time. For flight control systems, the timing situation is even more restrictive.

The definition of "late result" depends on the context and application of the real-time system. In general, from the perspective of the nature of the system deadlines, real-time systems are classified as soft real-time and hard real-time systems depending on the consequences of missing a deadline. Missing a deadline may not imply a system failure for a soft real-time system, but the usefulness of a result that misses its deadline is degraded. Thus, applications such as video decoders are soft-real time systems, in the sense that if a frame misses its deadline to be decoded, the image would not suffer much degradation in usefulness and the viewer may not notice the failure. In contrast,

for applications such as car's cruise control systems, anti-lock brakes, aircraft control systems and heart pacemakers, if they do not react in a certain strict period of time, the consequences could be catastrophic. These systems are hard real-time systems. Thus, for a hard real-time system it is imperative to guarantee the successful execution of every task before its deadline, whenever such tasks are activated.

Real-time systems are composed of collections of tasks that have specific timing and resource constraints that require the implementation of an efficient scheduling algorithm (or scheduling policy) to guarantee their successful execution under different timing constraints. The scheduler of any computing system decides what task to execute next when faced with a choice in the execution of a set of concurrent tasks. It also decides the assignment of resources to each task at any specific time [20]. A general purpose computing scheduler is aimed to schedule non-time-critical applications and may favor fairness of the distribution of resources among all tasks, as well as the overall throughput and performance, without considering strict finishing times for each task. The main goal when designing a real-time system is to guarantee the successful execution of all real-time tasks in the system before accomplishing the timing constraints. This makes the system's predictability, i.e. the capability to determine timing characteristics of a computing system with certainty, an important parameter to consider when designing a real-time scheduler.

Different real-time scheduling algorithms have been proposed to fulfill the different requirements of modern real-time applications. These scheduling algorithms can be classified in different ways, e.g. on-line/off-line, priority/non-priority, and single-core/multi-core [21].

Real-time scheduling algorithms can be implemented either as online or off-line mechanisms. Scheduling algorithms implemented online generate scheduling information while the system is running. They assume little or no a-priori knowledge of tasks that have not arrived yet, which restricts the potential for the system to meet timing and resource shar-

ing requirements. This type of algorithms are adaptive, flexible, and can consider in the scheduling decisions the run-time variations of the system and the environment. However, the calculations required to make such decisions may produce a large run-time overhead, which force them to implement simple scheduling rules. Also, this type of system may not offer guarantees to the system's timing constraints, due to its inherent variability nature. Unlike online scheduling algorithms, off-line ones generate fixed scheduling decisions prior to the system's execution. Such scheduling decisions are later utilized by a simple task dispatcher during runtime with a small run-time overhead. This property makes possible to implement off-line scheduling algorithms with high computational costs, because their high complexity does not affect the run-time overhead. By definition such algorithms are not flexible and usually must account for worst-case operating conditions, which may generate pessimistic schedules. In a predictable environment, off-line implementations can guarantee system performance, and most importantly, off-line scheduling algorithms can guarantee the system's timing constraints. These reasons make off-line scheduling algorithms a first choice for designing hard real-time systems.

One of the critical problems in real-time scheduling is in what order should the real-time tasks be executed. Depending on the properties of the real-time tasks or the scheduling implementation mechanisms, real-time scheduling algorithms can be categorized as non-priority-based or priority-based. In scheduling terms, a priority is usually a positive integer representing the urgency or importance assigned to a task. Thus, non-priority-based algorithms schedule tasks with the same level of importance, and priority-based algorithms make a distinction between the importance of each task.

Non-priority-based algorithms are suitable for applications requiring a guaranteed fair distribution of the resources among tasks. However, a fair distribution of resources may not be sufficient to guarantee the timing constraints, because the behavior of such scheduling methods is hard to predict. An example of non-priority-based algorithm is the Round-

Robin algorithm. This scheduling algorithm assigns a fixed amount of computation time to each task. This type of methods have the advantage that their implementation is relatively easy, compared to priority-based algorithms.

Real-time scheduling algorithms usually implement a strict order of execution for tasks in the system, in order to maximize its predictability. A common mechanism to develop such an order is by assigning tasks with priorities, which is the case for priority-based algorithms. By assigning tasks with different priorities of execution it is possible guarantee completion of tasks with higher time sensitivity over tasks with a lower time sensitivity. In some cases, the scheduling algorithm may allow preemption of tasks, so that the latency in executing higher priority tasks may be reduced by executing them over lower priority tasks. However, one of the severe problems that can occur with priority-based preemptive algorithms is the "priority inversion" phenomenon. Priority inversion may be present in systems executing tasks that have dependencies among themselves. Priority inversion occurs in a real-time system when a high-priority task has to wait for a lower priority task to execute, because another lower priority task is using exclusively a shared resource needed by the high priority task. Multiple improvement mechanisms and algorithms, e.g. the priority inheritance protocol (PIP), have been proposed in the literature to overcome this problem [22].

Multiple methodologies have been proposed for assigning priorities to each task either statically or dynamically. A very common method for assigning and scheduling fixed-priority tasks (statically) is the rate-monotonic scheduling (RMS). The RMS scheduling algorithm is one of the most widely studied and used in practice, due to its low overhead and simplicity in implementation [23]. This method assigns priorities to tasks in ascending order with respect to their periodicity; i.e., a larger task period leads to a lower priority task. RMS has been proven to be an optimal scheduling policy for fixed-priority tasks on a single-core processor [24]. A very common method for assigning and schedul-

ing dynamic-priority tasks is the earliest-deadline first (EDF). As stated in its name, this method assigns priorities to tasks dynamically in an inverse proportion with respect to the difference between the actual time and the task deadline. In other words, as the difference in current time and deadline time shrinks for a specific task, its priority increases. EDF has been proven to be the optimal dynamic scheduling algorithm for hard real-time tasks on a single-core platform [24].

Real-time systems used to be developed on single-core platforms because of the already high level of predictability achieved by single-core scheduling algorithms implemented in these types of systems. A high increase in processing demands has led the industry to develop them on multi-core platforms in the past few years. Thus, based on the underlying hardware infrastructure, real-time scheduling algorithms can be categorized as single-core scheduling algorithms or multi-core scheduling algorithms.

The design and implementation of an efficient scheduler for real-time tasks on a single-core platform is a hard work process, but scheduling tasks on a multi-core platform usually requires a bigger effort. Different from single-core scheduling, multi-core scheduling needs to decide not only when, but also where a task should be executed. Hence, multi-core scheduling is known to be a NP-hard problem and more complicated than single-core scheduling [14]. If tasks have dependencies, such as shared resources like memory, calculating task completion times on a multi-core system is inherently more difficult than on a single-core system.

Different multi-core scheduling mechanisms have been proposed, such as global scheduling, semi-partitioned scheduling, and partitioned scheduling. In the global scheduling approach, any job from any task can be executed on any processing unit of the system. For global scheduling algorithms allowing preemption, a single job may start its execution on one processing unit and resume on a different one. In the partitioned scheduling approach, it is fixed the processing unit that will always execute all jobs from a specific task. The

semi-partitioned scheduling approach is a combination of the two previous approaches, i.e. some tasks are assigned to a dedicated processor, while the rest of the tasks can be allocated among all available processing cores.

For the case of partitioned scheduling, by fixing the task to a processing unit, the predictability of the system is increased. This premise makes multi-core partitioned scheduling the first choice for designing a hard real-time system. However, the key problem becomes to statically choose what group of tasks allocate together, sharing the same system resources, e.g. sharing one or multiple cache memories, or one or multiple memory controllers. Thus, multiple research works have proposed various allocation strategies looking to improve the system schedulability, when scheduling tasks using multi-core partitioned scheduling. The most common studied ones are the strategies based on the traditional Bin-packing approach, i.e. First Fit (FF), Best Fit (BF), and Worst Fit (WF) [25].

Besides real-time constraints, real-time scheduling algorithms are also developed with different optimization goals in mind, such as power/energy consumptions, thermal impacts, reliability, etc. Therefore, real-time scheduling algorithms can also be categorized based on their design optimization goals as power/energy aware real-time scheduling, thermal-aware real-time scheduling, reliability-aware real-time scheduling, etc. Research works such as [26, 27] present power-aware and energy-aware schedulers that look to minimize the consumption of each parameter or both. Additionally, some real-time devices, either mobile or fixed, are intended to be used in enclosed extreme conditions or near environments that restrict their peak temperature of operation, beyond of the peak temperature operation imposed by the chip and its package. Therefore, some research works such as [28] have proposed schedulers to account for the power and thermal issues of the platform in order to guarantee an operation below a certain temperature threshold. Another example are devices implemented on computing platforms which have a great variability in their manufacturing characteristics from one processing unit to the next one.

Thus, some research works such as [29] have proposed schedulers that account for the changes in execution time due to the unexpected manufacturing process variations. Also, some real time devices are deployed on extreme environmental conditions that increases the probability of present computing errors at any specific time. Some research works such as [30] have proposed schedulers that implement supplementary timing error detection and correction mechanisms in order to maximize the reliability and fault tolerance of real-time systems.

To summarize, real-time systems are ubiquitous and critical to our daily lives. Real-time scheduling plays a critical role to ensure the timeliness of real-time systems, especially for real-time applications that are critically sensitive to time. In the past, numerous research works have focused extensively on real-time scheduling, but most of them have focused on the CPU resource management. This dissertation takes into consideration in the design process of hard-real time systems, not only performance characteristics of processing units, but also restrictions and latencies imposed by memory devices considering the effects that different memory parameters inflict on each real-time task.

1.2 The challenges presented by memory systems in design of real-time systems

Memory has played an important role in computers since the early days of computing. Memory is essential to the operation of a computer system because of its purpose of supplying instructions and data for calculations in a timely manner. In the early days of computing, accessing the memory was as fast as the rate of performing the actual calculations by the processor unit. As processing units became faster and as the size of problems grew, access to memory became steadily slower than the rate of computations [31].

Although a flat memory system built using a single manufacturing technology was desirable for implementing computing systems because of its simplicity, none of the available memory technologies is capable of complying with the continuously increasing need of having high memory speed, high memory capacity and low cost per bit. In this regard, nothing is more important to the development of modern memory systems than the concept of the memory hierarchy, for it provides a computing system with a memory closer to the ideal case of having the highest capacity, with the minimum latency for each memory access, at the lowest possible cost per bit [13]. However, memory hierarchy alone falls short of satisfying the rapidly growing needs of higher memory bandwidth and capacities for new generations of real-time applications. The execution time variances due to memory hierarchy also significantly degrade the predictability of real-time systems. Moreover, as memory capacity and transistor density continue to grow, the power/energy consumption and thermal impacts of memory devices also raise significant challenges in design of new generations of real-time systems.

1.2.1 The memory wall problem

Memory performance has also become a significant issue in the design of real-time systems. Real-time systems processing huge streaming of data such as cameras and specialized sensors are becoming popular, and such applications not only generate large amounts of I/O workloads, but also become more and more memory intensive, which is translated in the need to develop real-time platforms with higher memory bandwidth and capacity. Applications such as IoT, wearable, networking and automotive markets are driving main memory innovation. Fig. 1.3 shows the applications driving requirements envisioned by MICRON in the near future, where we can see that a variety of applications in embedded computing systems will rely on a higher memory capacity and bandwidth. Additionally,

Market	IoT	Wearables	Networking	Automotive
Application	MCU external memory	Graphics, Icons	Primary boot	Instrument cluster, ADAS, IVI
Performance Needs	Medium to High	Medium	Medium to High	Medium to High
Density Needs	Wide Range	High (128MB-1GB)	Medium (32-256MB)	Medium (32-256MB)

Figure 1.3: MICRON's applications driving requirements for embedded systems in the near future [4]

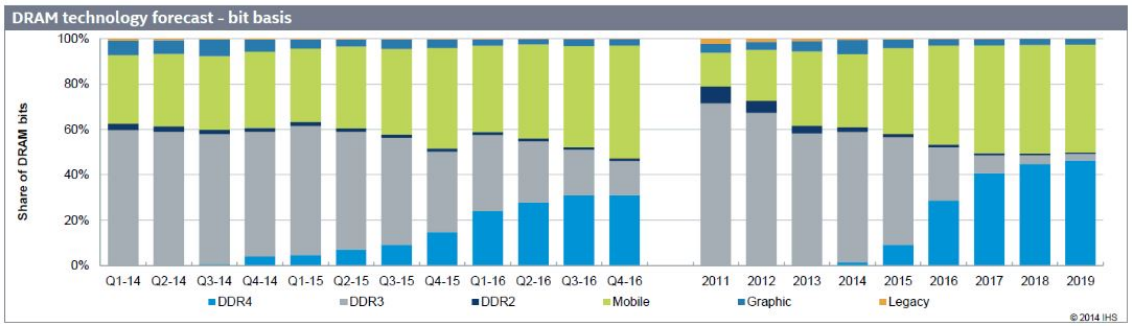


Figure 1.4: Technology forecast of share of DRAM bits [5]

as seen in Fig. 1.4 mobile DRAM share has been increasing over the last few years, which is an indicator of the increment of memory-bounded applications for embedded systems. As an illustration, in the automotive industry, there are now hundreds of microprocessors in every car. New applications such as the Advanced Driver Awareness Systems (ADAS), involving data processing for multicamera vision, improved infotainment, and even self-driving sub-systems, have demands for memory system inclusion of large capacity DRAM devices [32]. Specifically, a study shows that a network-connected car can create tens of megabytes of data per second and an autonomous vehicle is estimated to generate data at the rate of about 1 gigabyte per second [33].

Despite the fact that employing memory hierarchy can greatly improve the memory access performance, researchers noticed that the rate of improvement in microprocessor's speed exceeds the rate of improvement in memory speed. While each one is improving

exponentially, the exponent for microprocessors is substantially larger than that for memories. This disparity is known as the *Processor-Memory Performance Gap* or the *Memory wall problem* [34, 35]. In recent years, while the performance of processing cores has increased dramatically (60% per year), the improvement in access time of memory (10% per year) has not kept up with the pace [36]. According to Amdahl's law [35], the continuous increment in *Processor-Memory Performance Gap* would lead to a stall of system performance improvement no matter how much processor performance can be further improved. Therefore, how to overcome the so-called "memory wall" to satisfy the increasing performance demand of computing systems, including those of high performance real-time systems, has presented a great challenge problem, attracting tremendous research efforts from both industry and academy [31, 37].

1.2.2 The memory access time variation problem

In order to design an efficient scheduling algorithm for a specific real-time application, it is important to define the real-time system task model. A real-time task model refers to the set of pre-defined characteristics and assumptions that are fed to the scheduling algorithm, so it can make its decisions. Typically a real-time task is characterized by its execution time, period, and deadline. Unlike a typical general purpose computing scheduler, a real-time scheduler must assume that every task terminates, and such an event is associated with a predetermined deadline. Usually, it is also assumed that each task is repeated iteratively and a period time for the task execution is also assigned.

Along with the correct definition of a task model comes the correct estimation of the Worst-Case Execution Time (WCET) of each real-time task. The estimation of WCET values for real-time tasks is done either by performing a theoretical analysis of the machine code executed on a particular family of processor architectures with specific char-

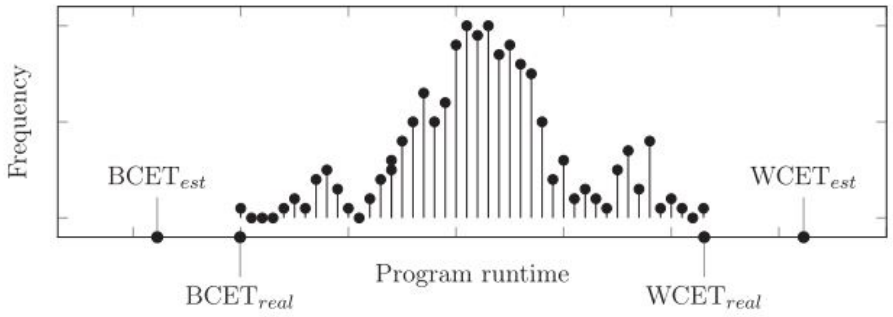


Figure 1.5: Typical sample distribution of runtimes of a program, along with sample of BCET and WCET [6]

acteristics, or by performing an empirical analysis where different execution times are measured considering different machine states and input conditions. Fig. 1.5 shows a typical sample distribution of runtimes of a real-time task, along with sample Best Case Execution Time (BCET) and WCET [6]. It can be seen that since the real WCET is unknown for many tasks, an often over-provisioned estimate must be assigned to each task, in order to develop a safe real-time scheduling.

The correct estimation of WCET values is critical in order to further estimate the amount of resources needed by any task when it is activated. An underestimated WCET may lead to assign a poor amount of resources to a task, which may signifies that such a task misses its deadline. This is why a correct WCET estimation is critical for hard real-time systems, because all hard real-time deadlines must be met under any circumstances. In contrast, pessimistic over-provisioned WCET estimates for some real-time tasks counteract the increments in performance expected from the platform deploying the real-time system. In other words, to excessively overestimate the WCET for each task, in order to obtain safe values, can have a negative effect and may nullify the extra computational capacity that the real-time platform may offer. Therefore, it is necessary to estimate an accurate and safe WCET for each real-time task to develop an efficient schedule that guarantees the successful execution of all real-time tasks [23].

The primary goal for any real-time scheduling algorithm is to complete all tasks within specific time constraints, by allocating the available resources of the system judiciously to each task. Although the maximization of system resource utilization is of interest, it is not a primary design motivation. In fact, as explained before, predictability and temporal correctness are the main concerns. Consequently, a problem driving the research community, during the past decades, is how to implement efficient schedulers able to achieve high computational performance while guaranteeing the timing constraints, and preserving the system predictability. For this purpose, it is necessary to estimate an accurate and safe WCET for each real-time task in the system, accounting for all possible execution time variations of each task. However, modern memory systems are introducing additional sources of execution time variation for real-time tasks. These execution time variations may result in direct predictability reduction for the whole system, and pessimistic over-provisioned WCET estimations.

A first source of execution time variation is due to a memory hierarchy implementing multiple levels of storage. In the vast majority of computing systems implementing a memory hierarchy, not every memory access from the same task has a uniform latency. This fact is mostly due to the the architectural features at each memory level, in combination with the task's characteristics (e.g. the task's instructions, variables, and data). For instance, a cache hit will have a different latency than a cache miss, and the number of cache misses may vary within multiple executions of the same task. Also, a memory access to a DRAM position may be prioritized differently depending on the region of memory it refers to. To put it differently, if the system would be executing a single task, such a task would generate a different memory access pattern every time it is activated. In essence, memory systems have been designed and commercialized to favor the reduction of latencies for the average case. Statistical observations show that memory hierarchy improvements speed up the computation time in average, by reducing the delay in memory

access time in average. However, statistical observations may provide only an estimation of the average behavior of a task, but they cannot be used for deriving worst-case bounds. Since safe WCET estimates come from an analysis that depends on architectural features, the memory hierarchy architectural advancements may lead to inaccurate or unsafe WCET estimations, degrading the predictability or the performance of the real-time system.

A second source of execution time variations is due to memory resource sharing among: (i) different real-time tasks, for either single-core or multi-core platforms; (ii) processing units, for the case of multi-core platforms. In preemptive systems, the memory systems are affected by the number of preemptions, because preemption nullifies the benefits of program spatial and temporal locality. For instance, a higher priority task may evict an unbounded number of cache blocks already brought to the cache by a lower priority task. The cache-related preemption delay (CRPD) depends on the specific point at which preemption takes place. Therefore such CRPD is very difficult to precisely estimate [14]. A similar effect is seen on DRAM memory systems implementing open-row policies. Multiple memory controllers implement complex algorithms to prioritize memory accesses referencing an already open row, because the access time to an already open row is much faster than to a closed one. For further details on the DRAM memory architecture, we reference the reader to the chapter 2 of this dissertation and to [13]. The unpredictability affecting real-time systems is exacerbated in multi-core platforms, because different tasks on different cores, contend for bandwidth and capacity at the different levels of the memory system, such as memory controllers, interconnects and caches. Thus, in multi-core platforms, the memory contention (i.e. memory interference) between different cores critically undermines the overall real-time system predictability, and therefore impacts even more its performance [38]. Large inter-task interferences due to increased resource sharing on multi-core platforms have severely undermined the predictability of

real-time systems [39,40]. An increase of 300% has been seen in the estimated values of WCET of real-time tasks, when memory interferences are taken into consideration [41], which can lead to extremely pessimistic designs.

The restrictions to real-time systems imposed by memory devices do not come exclusively from communication delays and timing related problems. Since the power and energy consumption of memory devices continue to grow, the effects in performance imposed by memory power and thermal management mechanisms are becoming more and more notorious. Thus, we also believe that it is necessary to address the scheduling restrictions inflicted on the execution of hard real-time tasks, added by power and thermal management mechanisms, when considering the power consumed by memory devices.

1.2.3 The power/energy consumption and thermal problem for memory systems

As transistor counts and density of processors and memory devices, as well as the memory capacity and bandwidth, continue to grow, the power consumption of computing systems has also been increasing exponentially, resulting in tremendous heat generation, even to the point that threatens to disrupt the operation of the system under normal conditions [42]. Fig. 1.6 shows the most important challenges for embedded systems development in 2015, and it can be seen that power management is one of the most important concerns with a consideration in 13% (compared to a 9% in 2014) of embedded system developments. An increasing chip temperature due to an excessive power dissipation has a significant impact on other design metrics, such as reliability, cost and especially on performance. Real time systems and specially embedded systems with real-time capabilities are developed on resource constrained platforms, which impose additional restrictions on how a real-time scheduler must manage the overall resources of the system in order to

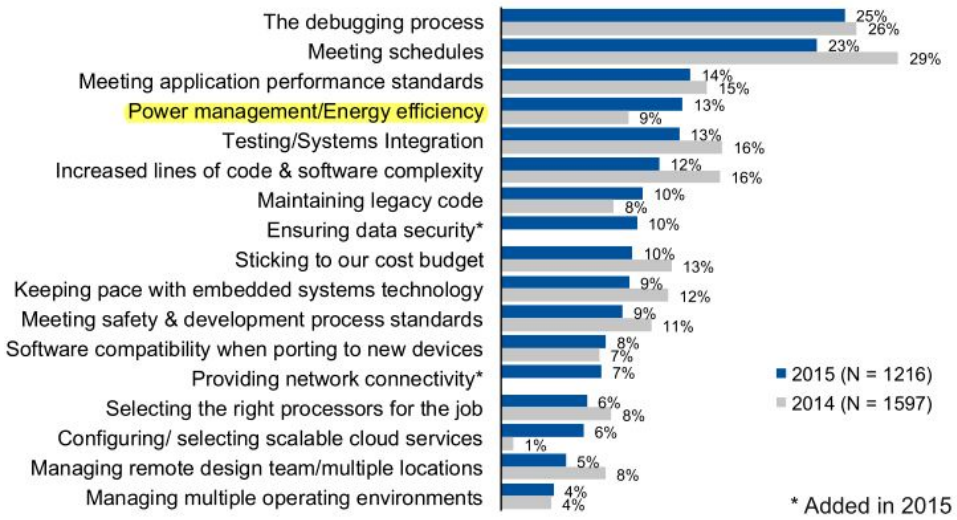


Figure 1.6: Most important challenges for embedded systems development in 2015 [3]

guarantee the timing constrains. An example of a constrained platform can be found in modern mobile systems, where it is important to maximize the operational autonomy of each device by operating them on tight power and energy restrictions, and it is important to keep temperature below unconformable threshold values for the final user. Furthermore, for multiple applications, cooling down the chip temperature using mechanical methods such as cooling fans, heat spreaders, and heat sinks becomes inadequate and too expensive. This is especially true, for instance, in platforms aimed for wearable devices, or for future generations of Internet-of-Things (IoT) applications [43].

Techniques implemented on processors based on the *dynamic power/thermal management (DPM/DTM)* mechanism have become an appealing solution to manage the thermal emergencies of the system. The DPM/DTM mechanism switch the processor to a low-power inactive state as long as possible, or following an specific power trace pattern, looking to reduce the overall energy consumption, or reduce the maximum system peak temperatures [44]. While many DPM/DTM solutions have been proposed in the literature (e.g. [45, 46]), most of them focus on considering the CPU characteristics exclusively, as CPU traditionally is the major power source in a computer system. However, the power

consumption of main memory has become a significant portion of total power consumption of the system, in processors ranging from low-end to high-end [18]. As an example, it is estimated that as much as 40% of the total power consumed in a smartphone or a data-center is attributed to its memory system [47–49], mostly the DRAM-based memory systems.

Systems utilizing modern versions of DRAM technologies suffer high power consumption if the performance needs are high, requiring appropriate power/thermal management mechanisms that consider the memory devices. Multiple versions and standards of DRAM technologies have been published, including standards for low-power energy devices. However, as mentioned before, all DRAM types of memories have experienced an immense bandwidth requirement increment and are expected to continue growing in density and performance. Fig. 1.7 shows the increment in data rate per pin on various types of DRAM memories in the past few years, and we can see that such rates have increased near six times for some DRAM technologies [7]. Additionally, Fig. 1.8 shows that DRAM technology density has increased by a thousand times over the past two decades, while its latency has decreased only by 56%, which brings design issues such as capacity/cost limitation, scaling, and severe die overhead increase. Another study [8] also shows that in contrast of the continued scaling of cost-per-bit, the latency of DRAM has remained almost constant for different DRAM standards up to DDR3. During an eleven-year interval, data in which DRAM's cost-per-bit decreased by a factor of 16, DRAM latency (as measured by the tRCD and tRC timing constraints –the two most important timing parameters when accessing DRAM memory –) decreased by only 30.5% and 26.3%, as shown in Fig 1.9. Hence, the reduction in power and increment in performance are always trade-offs for DRAM technologies. Although, the power efficiency of DRAM technology has decrease in the past decades (see Fig. 1.10), the number of DRAM chips necessary to achieve a high performance has increase considerably too. Thus, in Fig. 1.11 it can

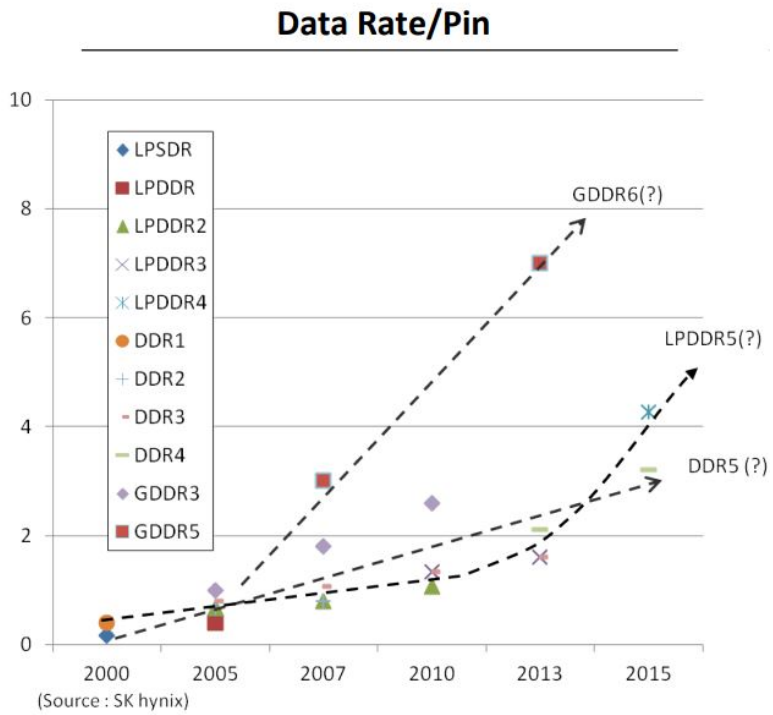


Figure 1.7: DRAM technology data rate per pin over the time [7]

be seen that to achieve a speed of 3.2 Gbps in 2017 it was necessary to have a DRAM memory with 78 DRAM chips and 1248 connection pins (i.e. # of DQ), compared to a speed of 0.8 Gbps, achieved with only 4 DRAM chips and 64 connection pins in 2008. The constant increment of DRAM chips leads to a continuously growing power and energy consumption in order to supply the increasing application's performance demands. Therefore, DRAM technology is known to have nowadays scalability problems, because its power consumption is reaching the system power threshold limits.

While some novel memory technologies [50, 51] help to reduce power consumption of memory chips, the small latency of DRAMs still makes them the top choice for main memory systems. Capacity, performance, scalability and also energy efficiency are the four key factors that designers of memory hierarchies have to deal with to satisfy the ever-increasing need for current data-intensive applications [52]. DRAM technology has faced difficulties in the scaling process in order to reduce its transistors size, and increase the

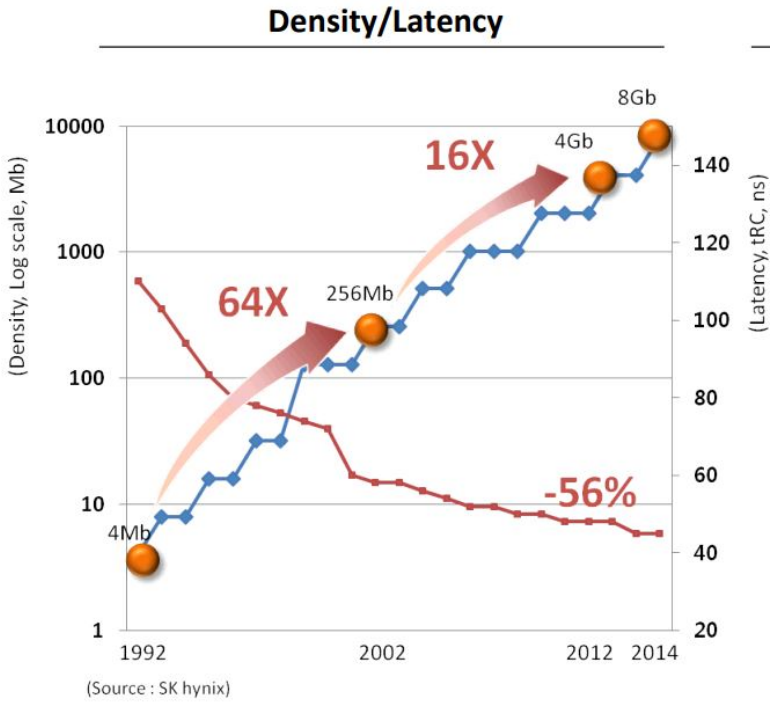


Figure 1.8: DRAM technology latency and density over the time [7]

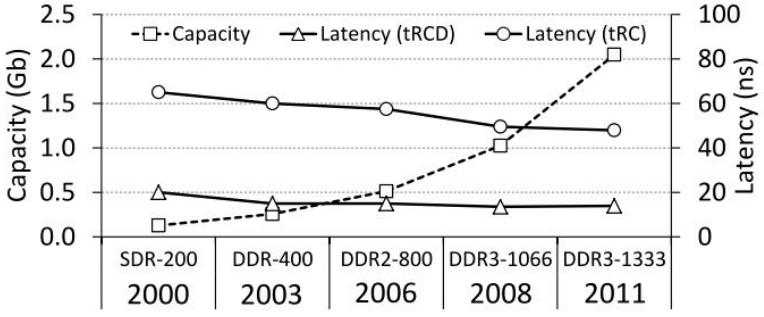


Figure 1.9: DRAM capacity and latency over the time [8]

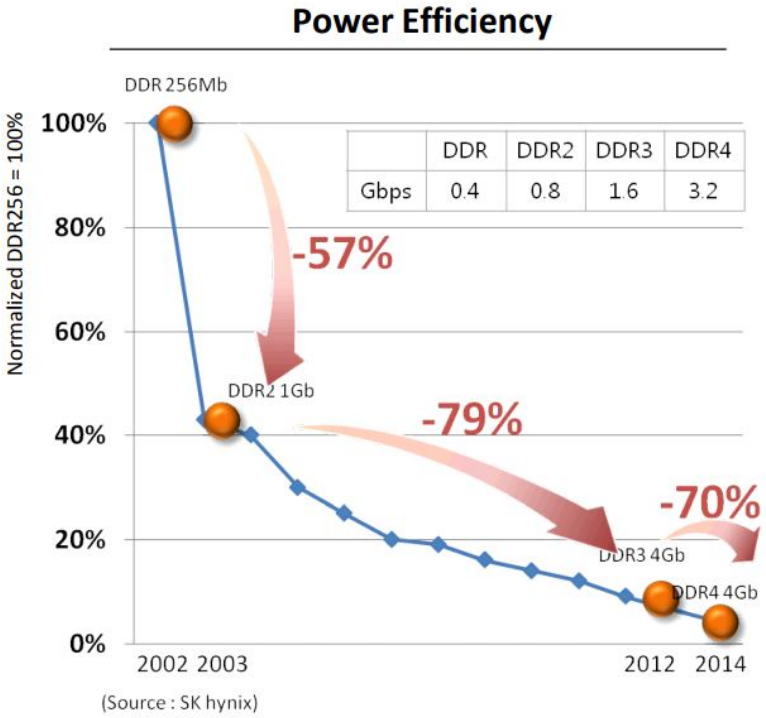


Figure 1.10: DRAM technology power efficiency decrement over the time [7]

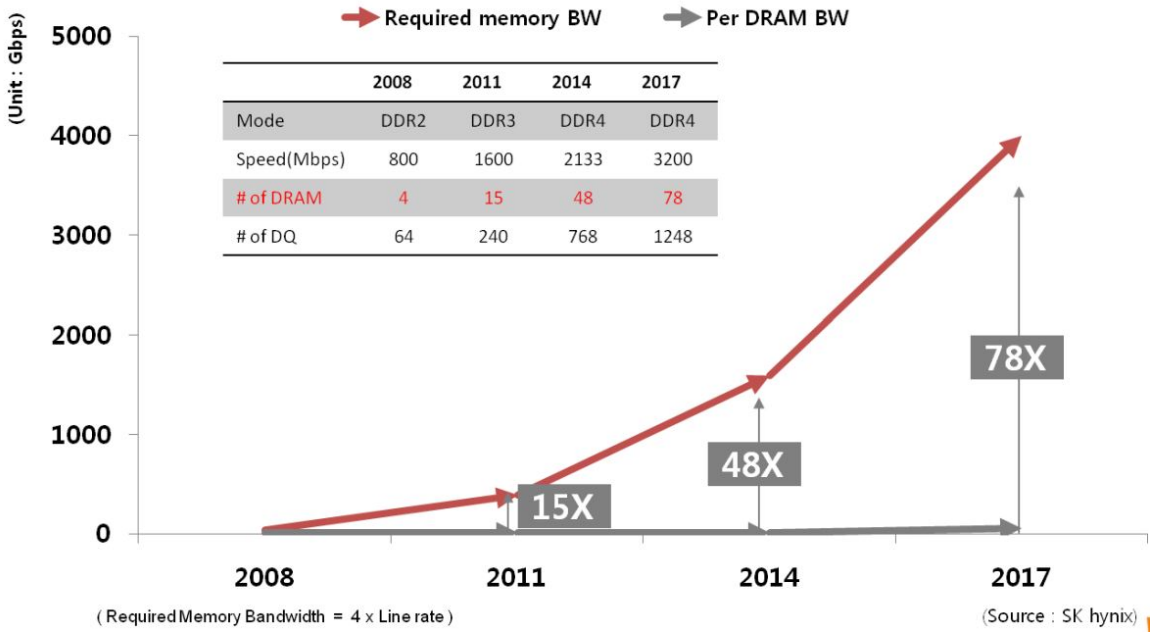


Figure 1.11: DRAM chip density increase over the time [7]

capacity of the memory. Besides, the inclusion of a larger number of transistors and cells is leading to an increment in power consumption due to the inherent leakage of current in each cell, which also increments the number of refresh cycles necessary to keep the memory properly functioning. Thus, a hypothetical 64Gb DRAM device would spend 46% of its time and 47% of all DRAM energy for refreshing its cells, in contrast to a 4Gb device which spend 8% of the time and 15% of the DRAM energy on refresh cycles [38]. There are multiple promising technologies to manufacture memory, such as Magnetoresistive Random Access Memory (MRAM), Phase Change Random Access Memory (PCM), Resistive Random Access Memory (ReRAM), and Ferroelectric Random Access Memory (FeRAM), each with its own peculiar properties and specific challenges. However, today there is still no memory technology able to surpass DRAM technology in all four key design factors of memory. Consequently, industry and academia have been looking to include different technologies into the memory hierarchy, in order to improve such factors along side with DRAM technology [52].

A high power consumption usually leads to high peak temperatures in any computing system. High temperatures decrease the DRAM retention capability, and increase the refreshing rates, impacting reliability, lifetime, power consumption, and specially system predictability [53, 54]. Additionally, it may also significantly degrade the performance and, therefore, compromise the timing constraints for real-time applications. For example, it has been reported [55] that for memory systems, when the temperature is above $60°C$, the performance/latency is degraded by the increment in the number of DRAM refreshes (around 50% more refreshes every $10°C$ of temperature increment), due to the leakage of charge on each DRAM cell [56]. DRAM high temperature problems are particularly exacerbated on multi-layer chips (2.5D and 3D chips) [57, 58] where restrictions in space and energy consumption are tight, power densities are higher, and CPU and memory temperatures are highly correlated. Most of today's CPU and memory chips are

embedded with built-in thermal sensors, and they will shut down automatically when the temperature exceeds a pre-determined threshold [59]. Such an unplanned shutdown will eventually cause real-time tasks to miss their deadlines.

As mentioned before, while several DTM-based solutions have been proposed, many of them have focused exclusively on CPU, as CPU traditionally is the major power source in a computer system. However, some of them have been aimed to dynamically manage the heat generated by the memory system, but without offering static guarantees on the scheduling of hard-real time tasks. For instance, one common approach is to migrate data between hot and cold devices to avoid thermal emergencies on a memory system [60]. Another approach dynamically adjusts the memory throughput to ensure that each module has a temperature below the emergency level [42]. These approaches do not take the heat generated by the CPU into account. Multiple DTM-based mechanisms already implemented in commercial hardware or proposed in the literature, react to critical temperature levels and reduce or even stall the number of memory requests [18]. Some research studies consider the heat dissipation from both the CPU and memory systems, but these approaches are best-effort approaches and cannot guarantee real-time system deadlines at all [61]. Consequently, such approaches can significantly affect the response time of the system by introducing an additional source of uncertainty, due to the variable number of stall times needed by the system to cool down the memory, each one with an unknown duration. Hence, the excessive power consumption and heat dissipation of the memory system must be dealt with carefully. Otherwise, they can significantly affect the schedulability and predictability of real-time systems because of the uncertainties introduced by different memory power/thermal management techniques.

In summary, a complete and effective thermal management solution, able to guarantee the schedulability of hard real-time tasks, should take into consideration not only the power and thermal issues of processing units, but also synergistically the power and ther-

mal issues of the CPU and memory subsystems as well. Studies have clearly shown that performance of memory systems (not only DRAM but other memory devices as well) is directly related with their operating temperature (e.g. [55]). In general, an increasing chip temperature, either on CPU or memory devices, due to an excessive power dissipation has a significant impact on other system design metrics, such as reliability, cost and especially on performance. Thus, the restrictions to real-time systems imposed by memory devices do not come exclusively from communication delays and timing-related problems. Since the power and energy consumption of memory devices continue to grow, the effects in performance imposed by memory power and thermal management mechanisms are becoming more and more notorious. For this reason, it is necessary to address the scheduling restrictions inflicted on the execution of real-time tasks, added by power and thermal management mechanisms, when considering the power consumed by memory devices.

1.3 The research problem and our contributions

A real-time system is a system whose execution time is expected to comply with deadlines, and missing a deadline is as negative as the incorrect output from the computation, leading to catastrophic consequences for certain applications. Thus, the most important requirement of a real-time system is predictability and not performance, which makes very important the correct implementation of a system scheduler to ensure timing constrains. Traditionally the scheduler design has considered only CPU and performance characteristics of the system. However, real-time systems processing lager amounts of data are becoming popular. Such applications not only generate large amounts of I/O workloads, but also become more and more memory intensive, which is translated to the need to develop real-time platforms with higher memory bandwidth and capacity. Hence,

the restrictions and latencies imposed by memory devices have gained a significant impact not only on the execution time, but also on other aspects such as power consumption and temperature of operation.

There have been extensive research efforts from different abstraction levels and perspectives, involving architectural hardware and software mechanisms, looking to improve the schedulability of real-time systems. For instance, since a major source of unpredictability when scheduling real-time tasks comes from shared cache memories, cache memory partitioning has proven to be one of the most effective methods to improve the predictability and schedulability of real-time systems. This method partitions cache memory among programs and cores to reduce cache contention. By isolating real-time task memory accesses, cache memory partitioning can avoid or considerably reduce the inter-task interferences, and therefore reduce the uncertainty when bounding the WCET and improve the core utilization [62]. Additionally, power and thermal management solutions at architecture and systems levels, such as dynamic thermal management (DTM) [18], and memory access throttling [63], have also been proposed to deal with power/thermal management-related uncertainties.

We seek to exploit these advanced features into a real-time scheduling framework to improve the feasibility in design of hard real-time systems. The challenge becomes how to incorporate these architecture and system mechanisms into one integrated framework, and to develop efficient and effective resource management solutions that can guarantee the timing requirements for hard real-time systems, maximizing the system schedulability, while also guaranteeing the operation under peak temperature constrains. This dissertation focuses on analyzing the problem how to design future hard real-time systems schedulers demanding highly deterministic computations when considering the role that memory systems play in designing effectively such hard real-time system scheduler algorithms. The general research problem can be formulated as follows: *Given a set*

of independent hard real-time tasks, implemented on a computing platform featuring a memory system, design static allocation scheduling algorithms to co-schedule CPU and memory subsystems, such that real-time and other design constraints (e.g. maximum temperature of operation) can be satisfied and other design metrics (e.g. feasibility or power/energy consumption) can be optimized. Toward this problem, we have made the following contributions.

1. First, we analyze the problem of how to assign private portions of cache memory, to real-time tasks, as a static memory resource management solution. We assumed a set of fixed-priority real-time tasks, to be scheduled on a multi-core platform that features a shared common cache memory. Our analysis is based on two facts: first, the performance of any task may be improved by increasing the size of the cache memory that task has access to, because the WCET of a real-time task varies depending on the amount of cache memory assigned privately to it; second, harmonic tasks can utilize the CPU resources more effectively, i.e. with a CPU utilization as high as 100% for each core in the system. However, not all the tasks can see the same amount of benefit in the reduction of its WCET by assigning them with an specific amount of cache memory. Thus, since this problem is known to be NP-hard, our approach seeks to develop a static task partitioning CPU and memory co-scheduling heuristic framework that wisely allocates tasks to cores, and portions of cache memory to tasks, synergistically, so that the feasibility of the whole system is increased, while guaranteeing hard-real-time deadlines. In essence, the proposed solution approach can judiciously choose the cache size for each task, while exploiting the task harmonic relationships within the task set. The significance of our proposed approach relies in that it enhances the existing heuristic allocation methods, by incorporating task period relation into cache allocation and task mapping, to improve the schedulability of hard real-time systems. Additionally, the proposed

solution approach statically co-schedule CPU and memory without increasing the predictability analysis complexity. The results show that the solution approach can significantly improve the schedulability of hard real-time tasks (up to four times), when compared with other scheduling mechanisms.

2. We also analyze the problem of how to guarantee timing constraints for hard real-time systems under CPU and memory thermal constraints. As previously explained, the increase in power density for both the CPU and memory systems makes necessary the implementation of effective thermal management mechanisms that can deal with the heat generated not only from CPU but also from memory. While many thermal management techniques have been proposed, most of them focus exclusively on either CPU or memory. Moreover, most of such techniques are on-line reactive in nature, which threatens the predictability of real-time systems. Our solution approach takes advantage of the periodic resource server for its capability of providing hard deadline guarantees for real-time tasks. The periodic resource server relies on the scheduling concept of providing real-time tasks with resources only during periodic windows of time, which increases the predictability of the system. Thus, by periodically (deterministically) throttling the accesses of the CPU and memory resources, our approach can effectively guarantee the thermal constraints for both the CPU and memory. To the best of our knowledge, this is the first work for thermal-aware hard real-time systems design that takes the heat generations and their interactions from both the CPU and memory devices. The significance of our proposed approach relies in that it enhances the existing thermal management methods, by incorporating the periodic resource sever model into a CPU and memory co-scheduling framework to guarantee both the timing and thermal constraints of the hard-real time systems. Our experimental results, with system parameters drawn from manufacture data sheets, clearly demonstrate the effectiveness of our

proposed approach in reducing the peak temperature, by supplying a static schedule that, combining active and power-down modes of each subsystem, generates deterministic power traces for CPU and memory devices. Additionally, such results show the need to take both the CPU and memory systems into considerations simultaneously for system-level thermal management.

3. Additionally, we analyze thermal-aware resource management strategies for both CPUs and memory systems when realizing hard real-time systems on 3D platforms under given peak temperature constraints. Designing 3D systems with on-chip DRAM is a promising solution to improve memory bandwidth and reduce memory access latency. However, 3D chips exacerbate the chip thermal problem due to their longer heat dissipation path, as well as the tight thermal coupling between logic and memory layers. Given the dramatically increased power density not only from CPUs but also from memory systems as well, we believe that a joint CPU and memory system resource management is highly desired for 3D platforms to effectively deal with the heat dissipation confined in a small package. In addition, different from many existing thermal management strategies, which are reactive and best-effort in nature, we are more interested in ones that can ensure the strong guarantee for real-time applications. Our novel solution approach also incorporates the periodic resource server to guarantee timing constraints for hard real-time systems under thermal constraints. We extended our analysis by proposing a solution approach for the case of multi-core architectures, when both CPU and memory devices are combined into a single package in a 3D integrated platform. The significance of our proposed approach relies in that it enhances the feasibility of existing heuristic multi-core task partitioning scheduling methods for real-time systems that require to be deployed under strict temperature constraints, by using thermal management mechanisms to co-schedule CPU and memory resources. Specifically, our

solution approach incorporates into the same scheduling framework the relationship among real-time task periods, the periodic resource sever model, and thermal analysis mechanisms for 3D integrated platforms. Simulation studies show that our proposed method can schedule on average 19.5% more tasks than the comparative methodology based on previous allocation mechanisms.

1.4 Summary and structure of the document

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 of how to allocate the cache memory that is accessible by multiple processing cores when scheduling fixed-priority real-time tasks based on the rate monotonic scheduling (RMS) policy. In Chapter 4, we study the problem of how to schedule fixed-priority real-time tasks such that they can meet their deadlines with temperatures for both the CPU and memory systems under their potentially different peak temperature limits. In Chapter 5, we study the problem of how to schedule a set of fixed-priority hard real-time tasks on a 3-D multicore platform, while keeping temperatures for both the logic layer and memory layer under peak temperature limits. Finally, in Chapter 6, we conclude this dissertation and discuss possible future work.

CHAPTER 2

BACKGROUND AND RELATED WORK

This chapter presents the pertinent research background and related work. We first introduce several important concepts related to real-time scheduling for single-core and multi-core platforms. Then, we introduce important concepts and related work about power aware/thermal aware scheduling. Further, we discuss the role, organization, and challenges of memory design and introduce important concepts and related work about memory aware real-time scheduling research. Finally, we summarize the contents of this chapter.

2.1 Real-time scheduling

Real-time systems are usually reactive systems that must comply with deadlines, and for such systems, missing a deadline is as negative as the incorrect output from the computation. The most important property of a real-time system is its predictability; that is, its functional and timing behavior should be as deterministic as necessary to satisfy system specifications. A correct real-time system must produce a functionally (algorithmically and mathematically) correct output response prior to a well-defined deadline relative to the request for a service [64].

A real-time system is often modeled as a finite collection of n independent recurring tasks τ, each of which generates a potentially infinite sequence of jobs [65]. The set of real-time tasks will be defined as $\Gamma = \{\tau_1, \tau_2, \tau_3, ..., \tau_n\}$. Each task τ_i is formally characterized by a worst-case execution time (WCET) requirement C_i, a relative deadline D_i and a period T_i. Such a task $\tau_i \in \Gamma$ generates a potentially infinite sequence of jobs, and successive jobs of τ_i arrive with at least T_i units of time apart. Thus, any task τ_i is usually represented by at least its timing parameters, i.e. $\tau_i = \{C_i, D_i, T_i\}$. In an implicit-deadline system, for each task τ_i, the deadline is equal to the period, i.e. $D_i = T_i$, $\forall \tau_i \in \Gamma$.

The main purpose of the scheduler in a computing system is to assign tasks to be executed by the processing unit in the case of a single-core platform or by each processing unit in the case of a multi-core platform. The scheduler usually manages processes or tasks with the concept of queues, i.e. long-term, middle-term and short-term queues. Such a scheduler selects tasks from each queue depending on the state of each task at any scheduling point in time (e.g. a task still waiting for a peripheral response cannot be scheduled to be executed yet). The specific schedule that may be generated by the scheduler is very important for any computing system. This is because it affects the overall performance of the system by determining which tasks will wait in the queues and which tasks will be executed.

Multiple types of general purpose schedulers have been developed and implemented on general purpose computing platforms. For instance, first-in-first-out (FIFO), round-robin (RR), or the completely-fair-scheduler (CFS). This last scheduler is implemented in the modern Linux kernel, and it is specifically aimed to maximize the overall utilization and performance of the whole system. However, unlike general-purpose schedulers, real-time schedulers are aimed to guarantee the timing and predictability of real-time tasks. This guarantee is performed basically by assigning a specific deadline for the successful execution of each real-time task, i.e. $\forall \tau_i \in \Gamma$, $\exists$ a time D_i such that$D_i \leq T_i$. Thus, such deadlines constitute the main difference between real-time and non-real-time scheduling. This is because any predetermined deadline should be met under all circumstances, even when the worst external conditions are present [66]. Therefore, the most important goal for a real-time scheduler is to maximize the system's predictability and not the system's performance, even to the point that by ensuring predictability, the whole system may decrease its performance.

For a real-time scheduler, it is necessary to ensure that resources are available for each task in time, and that the sequencing of events meets precedence constraints, in order to

guarantee all tasks' deadlines. A very common goal of a real-time scheduler is to allocate in the system as many tasks as possible, i.e. maximize the schedulability of the system, conditioned to meeting all tasks timing requirements. Satisfying the timing requirements of real-rime systems demands the scheduling of system resources according to some well-understood algorithms so that the timing behavior of the system is understandable, predictable, and maintainable. Therefore, every real-time system should implement a scheduler able to allocate resources judiciously to make certain that any critical timing constraints can be met with the available resources, assuming that the hardware and software function correctly and the external environment does not stress the system beyond to what it is desired to handle [67].

There are different ways to categorize real-time schedulers. In general, real-time schedulers can be categorized from the perspective of the characteristics of the system workload, the system architecture, the scheduling policy, and the different optimization objectives. In what follows, we discuss the details of the above categorizations to clearly understand the behaviors of real-time scheduling.

To begin with, real-time systems and schedulers can be classified according to the characteristics of the tasks they are intended to schedule. Thus, multiple schedulers have been developed to handle systems executing hard or soft real-time tasks, tasks with periodic or aperiodic behaviors, tasks with interdependencies on each other, or tasks with different levels of criticality within the same system.

Hard Real-Time and Soft-Real Time Tasks: The most common perspective classifies real-time systems depending on the consequences that may occur because of a missed deadline. Thus, a real-time task is said to be *soft* if any result produced after its deadline still has some utility for the system, but degrades performance. In contrast, a real-time task is said to be *hard* if producing the results after its deadline may cause a catastrophic consequence on the system. To avoid undesirable or catastrophic consequences, all hard

real-time tasks should be guaranteed on design time, i.e. off-line, and a hard real-time scheduler should aim to provide deterministic guarantees to all task deadlines. Examples of hard real-time tasks may be found commonly in safety-critical systems, typically related to sensing, actuation and control activities such as avionic systems [14]. In this dissertation, we focus our efforts on hard real-time tasks scheduling.

Periodic and Aperiodic Tasks: In a real-time system, a task can be periodic or aperiodic. A periodic task executes on a regular basis, and can potentially generate an infinite number of jobs activated or executed at a constant rate, which makes its execution deterministic. Aperiodic tasks, i.e. non-periodic tasks, also consist of an infinite sequence of jobs, but their activations are not regularly interleaved, which increases the unpredictability of the system. However, off-line guarantees for aperiodic tasks with critical timing constraints can be offered by making proper assumptions on the environment. Thus, jobs from aperiodic tasks can be assumed to be separated by a minimum inter-arrival time, and such tasks are commonly known as sporadic tasks [14].

Independent and Dependent Tasks: Different real-time schedulers are implemented with the assumption that all the tasks in the system are independent. In reality, it is possible to find that real-time tasks present dependencies. This means that one or more tasks in the system have explicit or implicit relationships specified among themselves. A common mode of dependency arises when one task needs the result of another one to proceed with its computations. For instance, in avionic systems, a task calculating positional error may need as input the result of another task calculating velocity and acceleration values. Also, certain tasks may require running in a certain order. For example, a module initialization task may need to be executed before other tasks may run.

Traditional scheduling mechanisms cannot directly be used to schedule tasks that share critical resources. Therefore, additional methods have been proposed to account for the additional restrictions imposed by the inter-dependencies of real-time tasks. How-

ever, traditional scheduling mechanisms, especially those assigning priorities of execution to real-time tasks (further explained in this section), can be augmented to make them applicable to tasks with dependencies [22]. In this dissertation, for the sake of simplicity and without loss of generality, we assume that all real-time tasks are independent in their execution. Any further mechanism able to account for tasks' dependencies can be applied orthogonally.

Mixed-Criticality-Based Scheduling: Real-time system have been diversified for multiple applications, generating a variety of real-time tasks with different properties. Other workload models, such as the mixed criticality-based scheduling, have been proposed aimed to schedule real-time tasks with different characteristics, timing properties and requirements. One common property considered to schedule real-time tasks is the criticality of each task. Criticality is the property that designates the level of assurance against failure needed for a system component. Thus, a mixed criticality system is one that has two or more distinct levels (e.g. safety-critical, mission-critical and low-critical).

A key aspect of mixed criticality real-time systems is that some system parameters, such as the WCET C_i of a real-time task, become dependent on the criticality level of the task. Thus, the same task will have a higher estimated value of WCET if it is defined to be safety-critical, than if it would be just considered mission-critical or non-critical. As the level of criticality changes during the operation of the system, the scheduler has to generate a different schedule, adjusting the values of the parameters of each task according to the current level of criticality set on the system. This behavior of mixed criticality systems can modify the results of traditional scheduling mechanisms [68]. However, the results of traditional scheduling mechanisms can also be augmented to make them applicable to tasks with different criticality levels.

Schedulers can also be classified according to the characteristics of the architecture of the system they are aimed to be implemented in. Thus, many schedulers were devel-

oped to schedule tasks on single-core platforms, but later different mechanisms have been proposed to schedule real-time tasks on multi-core platforms. Some of those multi-core platforms have been developed with homogeneous processing units, others with heterogeneous ones, depending on different applications. Thus, numerous real-time schedulers have been proposed for each type of platform. Also, either in single-core or multi-core platforms, multiple resources (e.g. peripherals, software routines, etc.) are limited in number. Therefore, scheduling mechanisms have been developed to guarantee the correct access of real-time tasks to such shared resources without affecting the strict timing behavior.

Single-Core and Multi-Core Scheduling: As mentioned in chapter 1 different mechanisms have been formulated for scheduling real-time tasks on single core platforms, but an increment in processing demands from real-time systems has led the industry to adapt some designs to be deployed using multi-core platforms. It is noteworthy to mention that a single-core system is built by integrating only one processing unit into a single chip, while multi-core systems integrate multiple processing units into the same chip. Different from single-core scheduling, multi-core scheduling needs to decide not only when a tasks needs to be executed, but also on what processing core. The two more general approaches to schedule tasks on a multi-core platform are *global scheduling* and *partitioned scheduling* [23, 69]. In general terms, the former treats every task and processing unit equally and each task that is ready to start or resume its execution is assigned to any processing unit available in the system. Note that a task may start executing on one processing unit and resume in a different one. Partitioned scheduling allocates tasks to be scheduled within a specific processing unit. Thus, each processing unit of the multi-core platform will be assigned to execute exclusively a specific sub task set. Only such a sub task set will share the resources assigned to its processing unit.

With a static multi-core partitioned scheduling, the problem becomes how to allocate statically real-time tasks to cores in a way to maximize the schedulability of the whole task set, while guaranteeing the timing and resource constraints of the system. Multiple solutions to this problem have been proposed by many authors in the literature. For instance, a common implemented solution is to allocate tasks on each core of the multi-core platform using partitioned scheduling, incorporating different allocation schemes such as traditional bin-packing approaches, i.e. First Fit (FF), Best Fit (BF), and Worst Fit (WF) [25]. Then, a specific intra-core scheduling policy to schedule the already assigned tasks to the core can be used. Other solutions, such as the ones presented in [70], also have evaluated how the ordering of tasks can affect the task allocation results, proposing additional scheduling frameworks using partitioned scheduling.

Homogeneous and Heterogeneous: Further, multi-core scheduling can be classified as *homogenous* and *heterogenous* according to the characteristics of the underlying multi-core system. In homogenous systems, all processing cores are identical in terms of processing speed, power/thermal characteristics, and so forth. In comparison, the cores in a heterogenous system can vary widely, which further complicates the multi-core scheduling problem.

Schedulers can also be classified according to the characteristics of the scheduling policy they will be implementing. Thus, many scheduling policies can be categorized as static or dynamic approaches. Certain policies may assign priorities to the real-time tasks, while other policies may not. Also, some policies may allow preemption of the scheduled tasks, while other policies may prevent such a behavior.

Static and Dynamic: In the past, numerous schedulers have been proposed to guarantee the timing constrains of real-time tasks. Depending on the type of real-time system, some of those schedulers have been developed to schedule tasks statically, i.e. in a fixed predetermined way; others have been developed to schedule tasks following a set of rules,

but able to make decisions dynamically during the process of scheduling tasks. On one hand, scheduling decisions for each task made by a static scheduler need to be predetermined beforehand, which requires prior knowledge of the characteristics of the tasks, the system and its environment. On the other hand, a dynamic scheduler needs to perform such decisions on-line, based on calculations performed during runtime. Thus, such schedulers can provide more flexibility to react to uncertainties of tasks and system resources, as well as environment conditions. However, it is difficult to provide strict timing guarantees with a dynamic scheduler [23]. This makes static schedulers the first option to implement a highly time-sensitive real-time system, such as hard real-time ones.

Preemptive and Non-Preemptive: On one hand, a scheduler that allows all tasks to run until completion is known as a non-preemptive scheduler. The design and analysis of such types of schedulers are relatively simple and have served as basis for multiple proposed scheduling mechanisms for real-time systems. On the other hand, a preemptive scheduler is able to make scheduling decisions during the execution of any task, stopping the current executed task and assigning any other ready task to be executed. The operation of suspending the running task and inserting it into a ready queue is called *preemption*. Preemption in real-time systems is important because it allows exception handling tasks to be executed over the current running task, allowing the scheduler to execute the critical tasks as soon as they arrive. However, preemption destroys program locality and introduces a runtime overhead that inflates the execution time of tasks [14].

Priority-Driven and Non-Priority-Driven: One of the major differences between traditional computing scheduling and real-time systems scheduling focuses on how to schedule recurrent jobs from different tasks. Multiple methodologies have been proposed in real-time scheduling in order to offer certain guarantees on the successful execution of each incoming job. For instance, a round-robin scheduling would be suitable for some applications in order to guarantee a fair distribution of the resources among tasks and

multiple works have been proposed based on such a mechanism. Such works treat each task without any specific priority and are known to be non-priority-driven. However, a fair distribution of resources may not be sufficient to guarantee the timing constraints of some real-time tasks in the system, because the timing behavior of each task is hard to predict with such scheduling methods.

Multiple scheduling methods that assign priorities to real-time tasks have been proposed and analyzed in order to offer timing guarantees and analysis methods to verify the schedulability of a real-time task set. In scheduling terms, a priority is usually a positive integer representing the urgency or importance assigned to a task. The ability to assign a priority of execution to a task is specially important to schedule hard real-time tasks sets due to the strict determinism they require, because a high priority task will have a higher chance to utilize the system resources before than a low priority task, and finish before its deadline.

Two single-core priority-based preemptive scheduling policies, i.e. Rate Monotonic Scheduling (RMS) and Earliest Dead-line First(EDF), have gain a especial interest in the research community and in the industry. These two scheduling policies play a fundamental role in the design of real-time scheduling algorithms [71, 72].

Rate Monotonic Scheduling (RMS) and Earliest Deadline First Scheduling (EDF): A very common method for assigning and scheduling priority-drive tasks in a static fashion way is the Rate-Monotonic Scheduling (RMS). This method assigns priorities to tasks in ascending order with respect to their predetermined period T_i, which means that a larger task period leads to a lower priority task, i.e. if $T_i < T_j$, then priority of task τ_i is higher than priority of task τ_j. RMS has been proven to be an optimal scheduling policy for fixed-priority tasks scheduling on single-core processors [24].

A very common mechanism for assigning and scheduling priority-driven tasks in a dynamic fashion way is the Earliest Deadline First Scheduling (EDF) method. As stated in

its name, this method assigns priorities to tasks dynamically depending on the remaining time for each task completion. Thus, as the difference in current time and deadline time shrinks for an specific task, its priority increases. This method evaluates and assigns the priority of each real-time task, every certain period of time during run-time. EDF has been proven to be the optimal dynamic scheduling algorithm for hard real-time tasks on a single-core platform [73].

The feasibility of a task set can be defined as the successful execution of all real-time tasks before their deadlines. Such a feasibility should be guaranteed in advance; that is, before task execution. Thus, it is necessary to perform an off-line mathematical analysis of how precisely the system timing constraints are met, to guarantee the feasibility of the whole real-time task set, implementing a certain scheduling algorithm. Such mathematical analysis is called schedulability analysis. It must consider both the task model parameters predetermined for each task, each task assigned priority, and the available resources in the system. If more flexibility is needed, for instance in a soft real-time system case, additional best-effort scheduling techniques can be applied, limiting the number of deadlines missed. Multiple sufficient schedulability conditions have been proposed and demonstrated in the literature in the past decades. The most common one was proposed by Liu and Layland [24], stating the task set CPU resource utilization upper-bound when scheduling tasks using RMS. However, a higher task set utilization can be guaranteed by an appropriate choice of task periods.

Harmonic-Periodicity-Based Scheduling Model: Some characteristics of real-time tasks have been exploited to develop more effective multi-core task partitioning schemes [1, 30, 74–76]. For instance, as shown in [1], by grouping harmonic tasks into the same core, system schedulability can be greatly improved, with each core reaching a much higher processor utilization than that defined by the RMS utilization bound, as defined in [24]. Harmonic tasks are defined as tasks with periods being integer multiples of each

Table 2.1: Example of task set to be scheduled in two processing units using RMS [1]

Task Number (τ_i)	**WCET** (C_i)	**Period** (T_i)	**CPU Utilization** (U_i)
1	1	4	0.25
2	2	8	0.25
3	3	10	0.30
4	8	16	0.50
5	8	20	0.40
6	8	40	0.30

other. If task periods are harmonic, i.e., each task periodicity value is a exact integer multiple of the next task periodicity value, then the schedulability can be guaranteed up to a 100% of processing unit utilization [75, 77]. Table 2.1 shows an example of a task set to be allocated in two processing units (core 1 and core 2). The task number is correlated to each task's priority, i.e. for τ_i and τ_j, if $i < j$ then $Priority(\tau_i) > Priority(\tau_j)$. It can be seen that both processing units can achieve a 100% utilization, by choosing the correct mapping of tasks allocated to each core. Therefore, fig. 2.1 shows the timing feasibility check for each processing unit scheduling tasks using RMS when tasks 1, 2 and 4 are statically assigned to core 1, and tasks 3, 5 and 6 are assigned statically to core 2. Partitioning of dynamic-priority periodic tasks on multi-core processors has been explored as well [78]. Other research works have proposed task partitioning schemes for hard real-time tasks with fault-tolerance requirements on multi-core platforms by exploiting the periodic relationships among tasks [30].

Sever-Based Scheduling Model: Real-time systems are often complex systems that must react to the environment. This premise is especially true for some embedded systems. These kind of systems commonly deal with tasks whose activation is triggered by sensor readings or voltage signal changes. Since the activation of such tasks is non-deterministic, those tasks are considered non-periodic tasks, i.e. sporadic tasks. Sporadic tasks are commonly required to be scheduled in real-time systems. Numerous real-time systems aimed for control applications require scheduling both types of processes, i.e. pe-

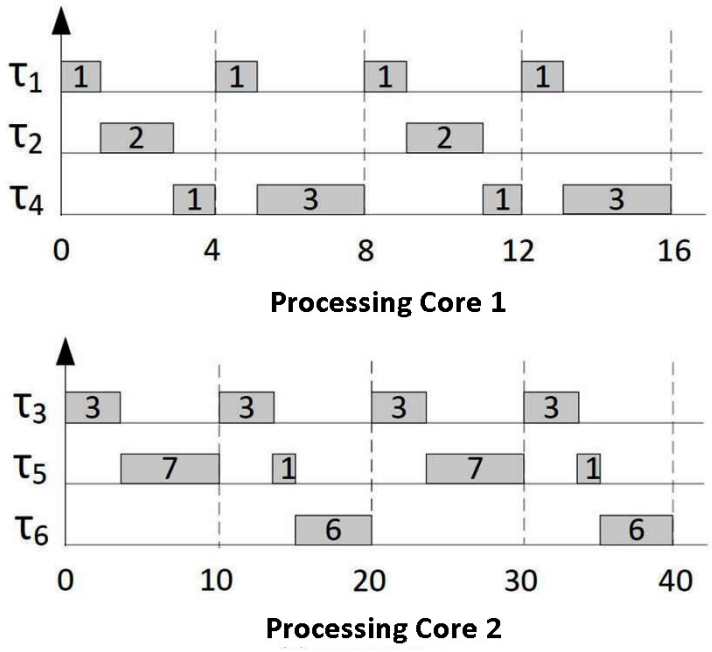


Figure 2.1: Example for timing feasibility check for each processing unit scheduling tasks using RMS (see Table 2.1), when tasks 1, 2 and 4 are statically assigned to core 1, and tasks 3, 5 and 6 are assigned statically to core 2 [1]

riodic and non-periodic. For any real-time task, the characteristic of being either periodic or non-periodic may differ from its criticality. Again, off-line guarantees for aperiodic tasks with critical timing constraints can be offered under peak-load situations, by assuming each job to be separated by a minimum inter-arrival time. This analysis is important to design schedulers able to schedule non-periodic hard real-time tasks.

If the minimum inter-arrival rate of a sporadic task cannot be bounded in design time, such an aperiodic task cannot be guaranteed off-line. However, an online guarantee of individual aperiodic requests can still be done. Aperiodic tasks requiring an online guarantee on individual instances are called firm real-time tasks. Whenever a firm aperiodic request enters the system, an acceptance test can be performed by the scheduler to verify whether the request can be served within its deadline. If such a guarantee cannot be done, the request is rejected.

Multiple scheduling algorithms for handling hybrid task sets consisting of a subset of hard periodic tasks and a subset of soft aperiodic tasks have been proposed in the literature and implemented on different real-time systems. Such solutions are based on the concept of *server*; that is, a periodic task characterized by a period T_s and a computation time C_s called *capacity* or *budget*. The server is scheduled with the same algorithm used for the periodic tasks, and once activated, it serves the aperiodic tasks within the limit of its budget. The scheduling of aperiodic tasks within the server can be performed independently of the global scheduling algorithm in charge of scheduling periodic tasks [14]. This concept will be useful to explain further the concept of compositional real-time scheduling, and how to offer guarantees to periodic tasks associated with a server instance.

Compositional Real-Time Scheduling: An important alternative concept proposed in the literature towards the analysis and scheduling of periodic and aperiodic real-time tasks is the possibility to allocate system resources privately to a specific subset of tasks within a certain periodic time. Such a concept would enable the possibility to guarantee the schedulability of a subset of real-time tasks with similar characteristics, and to guarantee the schedulability of each subset independently from the schedulability of other subsets. For this, it is necessary to implement a closed environment able to supply private resources to each subset of tasks. This environment may be associated to a periodic *server* task, with each periodic server having a period and a budget (also known as allocation time). Any task associated to the server may be executed only using the resources allocated to the server within the specified allocation time of the server [14].

Multiple research works have been proposed based on the *server* concept but one in particular was proposed by Shin and Lee [79] [9]. Their research work claims that for real-time systems, it is desirable to design large complex systems by breaking them into simpler components, based on systematic abstraction and composition. In this way, real-time tasks with similar characteristics can be abstracted into a single subset of tasks

forming a single component. The primary goal of these authors is to develop a compositional real-time scheduling framework to support abstraction and composition techniques for real-time aspects of components. A central idea in component-based design is to assemble components into a system without violating the principle of compositionality such that properties that have been established at the component level also hold at the system level. Especially for real-time systems it is necessary to support the abstraction and composition for timing properties of the system. Thus, they introduce the "periodic resource server model" to characterize resource allocations provided to a single component. Their work presents the exact schedulability conditions for the standard Liu and Layland [24] periodic task model when applied to the proposed periodic resource server model under EDF and RMS scheduling. Our work for this dissertation represents an advancement of the state-of-the-art for scheduling hard real-time tasks, because it incorporates the periodic resource sever model concept into the scheduling framework to guarantee both the timing and thermal constraints of real-time systems when considering the memory power and timing characteristics.

An example of hierarchical scheduling framework proposed by Shin and Lee is shown in Fig. 2.2, where it can be seen that different portions of a resource are scheduled by a high-level scheduler and each share of the resource is subsequently scheduled by a different scheduler. Thus, if a periodic server is seen as a resource, a group of tasks may be assigned to such a server, and scheduled with their own intra-server scheduling policy (such as RMS or EDF). Then, each server may be treated as a periodic task by a higher abstraction layer scheduler. Therefore, each periodic resource server on each abstraction layer may have a different scheduling methodology to schedule its own tasks.

The periodic resource server model proposed by Shin and Lee is characterized by a 2-tuple (Π, Θ), where Π is the resource period, and Θ is the allocation time. Both parameters satisfy that $0 < \Theta \leq \Pi$. The capacity of each periodic resource server is

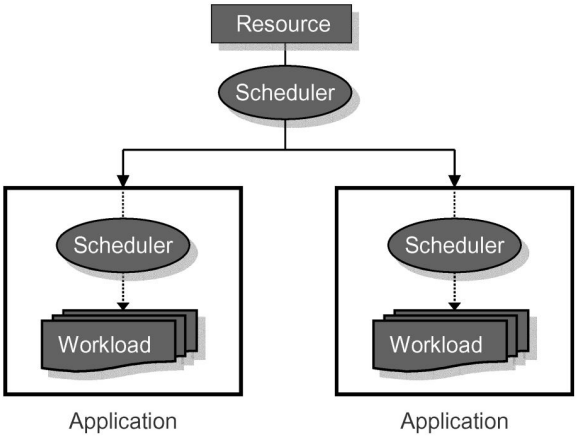


Figure 2.2: Hierarchical scheduling framework: a resource is scheduled by a scheduler and each share of the resource is subsequently scheduled by a different scheduler [9]

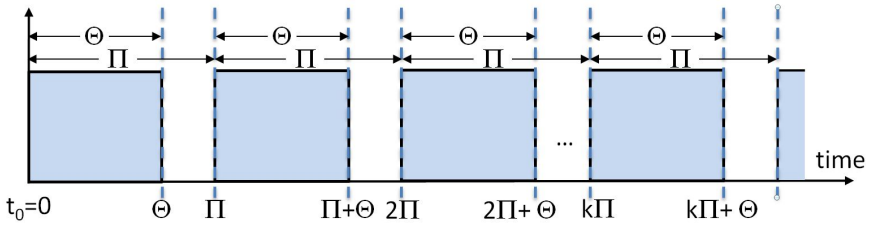


Figure 2.3: Shin and Lee [9] periodic resource server example

defined as $\mathbb{C} = \Theta/\Pi$. Also, the main characteristic of the periodic resource is that each task assigned to each periodic resource is allowed to be scheduled, executed and consume resources only during the allocation time Θ. Figure 2.3 shows an example of the periodic resource model.

Shin and Lee defined a utilization upper bound to guarantee the schedulability of real-time tasks assigned to be executed within a periodic resource server when the intra-server scheduling method is RMS, i.e. UB_{RMS}. Thus, if the total task set utilization is less than or equal to the upper-bound, such a task set is schedulable according to RMS, i.e. if for a specific Γ, $U(\Gamma) = \sum_{i=1}^{n} C_i/T_i \leq UB_{RMS}$, then Γ is schedulable according to RMS. Such upper-bound is defined in the following theorem:

Theorem 2.1. *[9, 80]. Given a task set Γ and a single periodic resource server, with period Π, allocation time Θ, and capacity $\mathbb{C}$, i.e. $\mathbb{C} = \Theta \div \Pi$, if $\forall i, 1 \leq i \leq N, T_i \geq 2\Pi - \Theta$, the task utilization bound under RMS is:*

$$UB_{RMS} = \mathbb{C} \cdot N \left[\left(\frac{2k + 2(1-\mathbb{C})}{k + 2(1-\mathbb{C})} \right)^{1/N} - 1 \right], \tag{2.1}$$

where $k = max\{k \in \mathbb{N}^0, s.t.(k+1)\Pi - \Theta < T_{min}\}$, with $T_{min} = min\{T_i | \forall \tau_i \in \Gamma\}$.

Also, Shin and Lee proposed the concept of *Abstract Bound* under RMS (AB_{RMS}). The abstract bound under RMS of any periodic server is the minimum server capacity $\mathbb{C}$ that such a server must offer to its assigned tasks in order to guarantee their schedulability under RMS. Such abstract bound is defined in the following theorem:

Theorem 2.2. *[9]. Given a task set Γ and a single periodic resource server, with period Π, allocation time Θ, and capacity $\mathbb{C}$, i.e. $\mathbb{C} = \Theta \div \Pi$, if $\forall i, 1 \leq i \leq N, T_i \geq 2\Pi - \Theta$, the server abstract bound under RMS is:*

$$AB_{RMS} = \frac{U_\Gamma}{log \left\lceil \frac{2k + 2(1-U_\Gamma)}{k + 2(1-U_\Gamma)} \right\rceil}, \tag{2.2}$$

where $k = max\{k \in \mathbb{N}^0, s.t.(k+1)\Pi - \Theta < T_{min}\}$, and U_Γ is the total CPU utilization of task set Γ.

Different optimization criteria: Some research works have proposed schedulers that by considering different applications and platform restrictions, they look to optimize different aspects of the system such as power consumption, thermal chip operation, reliability and fault tolerance. An example of a constrained platform can be found in modern mobile systems, where it is important to maximize the operational autonomy of each device by operating it on tight power and energy restrictions. Research works such as [26, 27] present power-aware and energy-aware schedulers that look to minimize the

consumption of each parameter or both. Additionally, some real-time devices, either mobile or fixed, are intended to be used in enclosed extreme conditions or near environments that restrict their peak temperature of operation, beyond the peak temperature operation imposed by the chip and its package. Some research works such as [28] have proposed schedulers to account for the power and thermal issues of the platform in order to guarantee an operation below a certain temperature threshold. Another example is devices implemented on computing platforms, which have a great variability in their manufacturing characteristics from one processing unit to the next one. Thus, some research works such as [29] have proposed schedulers that account for the changes in execution time due to the unexpected manufacturing process variations. Also, some real-time devices are deployed on extreme environmental conditions that increase the probability of present computing errors at any specific time. Some research works such as [30] have proposed schedulers that implement supplementary timing error detection and correction mechanisms in order to maximize the reliability and fault tolerance of real-time systems.

Multiple research works predominately focus on guaranteeing the timing constraints for hard real-time tasks based on the characteristics of the processing unit(s). As discussed in Chapter 1, other design constraints, e.g. power consumption and temperature of operation, are becoming increasingly critical in the design of real-time systems. Specifically, our interest for this dissertation is to analyze how to schedule fixed-priority hard real-time tasks with memory impacts taken into consideration also as design constrains. In what follows, we introduce some important real-time scheduling techniques that explicitly account for these design constraints, and also relevant concepts related to the co-scheduling of CPU and memory.

2.2 Power/thermal-aware scheduling

Due mainly to cost issues, the trend for many years was to develop computing platforms with a single-core chip and increase its performance by increasing the number of transistors and the frequency of operation of the chip, but the industry realized that with an increase in transistor density, the power density of the chip increases as well, making the power consumption of a single chip excessively large, requiring cooling capabilities beyond reasonable techniques. Fig. 2.4 shows the power density (*a*) and total power consumption per chip (*b*) over the time. It can be seen in Fig. 2.4*a* a steady increment of power density in PC microprocessors trend until 2006, when Intel released a chip with a power density higher than the core of a nuclear pressurized water reactor (PWR). After that, chip design strategies changed, including the popularization of multi-core platforms, looking to reduce the high-power densities. By adopting multi-core architectures, the potential performance gains due to parallel execution of tasks were increased, making the chip performance rely less on the increment in clock frequency. Multi-core architectures also offered the possibility of utilizing only some portions of the chip, which helps to reduce the power and energy consumed. Fig. 2.5 shows a significant increment in typical microprocessor chip power consumption and their operating frequencies over the past 40 years, but also shows both parameters being relatively steady during the past few years. However, looking at the absolute power consumed by a microprocessor chip (Fig. 2.4*b*), it indicates that with the increasing number of transistors (see also Fig. 2.5), the total power of microprocessor units is still increasing over time despite the reduction in power density [10].

In order to design a computer platform, multiple design constraints and specifications must be taken into consideration. However, nowadays a big portion of design time is tailored to manage the power and thermal issues of computing systems, because multiple

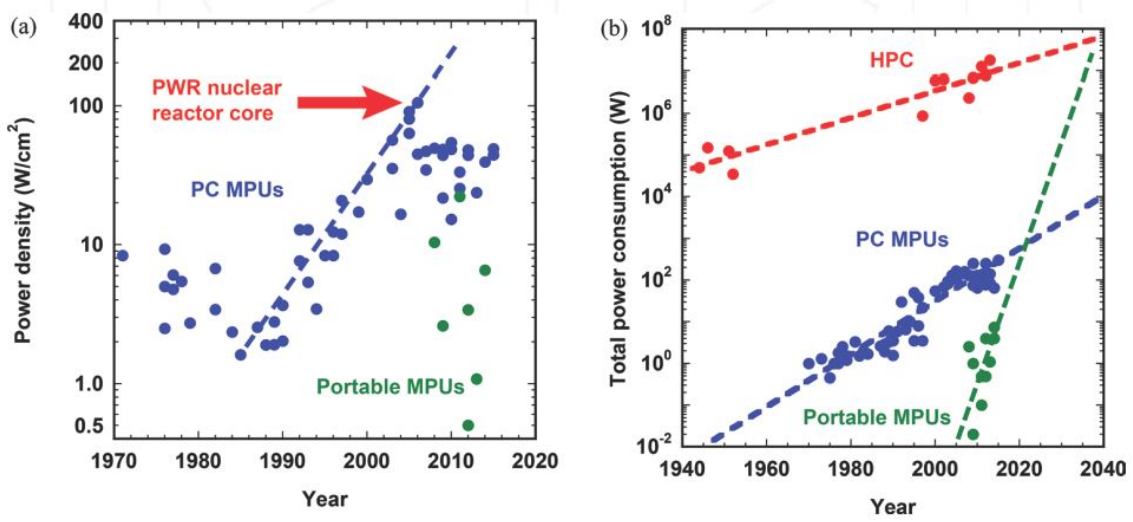


Figure 2.4: Power density and total power consumption of computing platforms over the years [10]

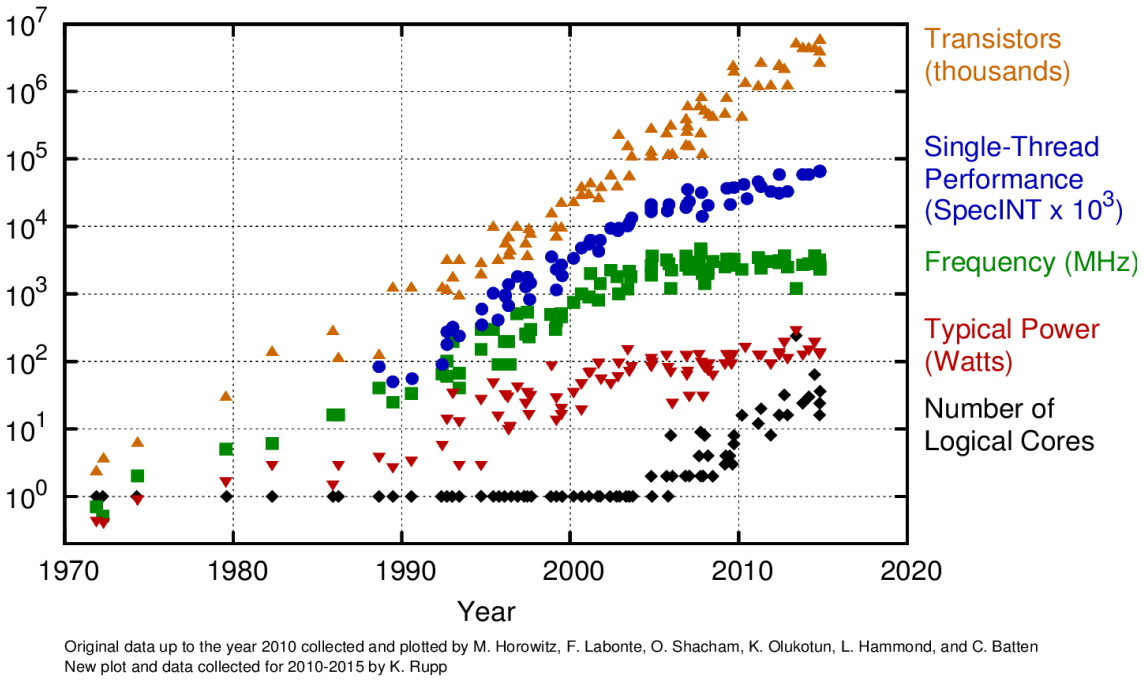


Figure 2.5: 40 years of microprocessor trend data [11]

problems in computing platforms are derived from a high power and energy consumption, including an excess in temperature operation of such devices. For instance, the increase in temperature leads to an increase in leakage power in transistor-based computing systems, which exacerbates even more the problem of excessive power consumption. Also, the increment in power consumption goes in detriment of the autonomy of battery-powered applications such as IoT devices, that need not only a miniaturized and autonomous platform to be portable and self-sustainable, but also to be energy-efficient as well. Additionally, high power consumption increases dramatically the maintenance cost of high-performance computing systems. Future exascale computers capable of reaching 10^{18} operations per second will require a substantial decrease in the amount of power and energy dissipated into heat compared to present standards. For instance, the U.S. Department of Energy aims to bring its first exascale supercomputer on-line sometime in the 2020s and their goal is to have the machine consuming no more than 20 megawatts [81].

As power and energy consumption has emerged in previous years as a critical design concern when designing computing systems ranging from high-performance computers (HPC) to embedded systems, it is expected to continue being an important aspect in the design of computing systems in the future. Thus, numerous research projects have proposed techniques to perform power management and thermal management in computing platforms. Some of those works are aimed to propose mechanisms to manage the power and thermal issues in HPC systems [82–85], while other research works [26, 49] are tailored to manage power and thermal on lower performance applications and embedded systems. Most real HPC applications do not utilize the peak power allocated power-per-node, leading to inefficient use of both nodes and power. Thus, in average, applications utilize 70% or less of the provisioned power, which leads to an inevitable waste of not only power, but also performance and infrastructure, making clear that hardware solutions are not sufficient and improved software solutions are needed as well for power manage-

ment [86, 87]. Additionally, modern supercomputers consume an enormous amount of power, where a significant fraction is dedicated to offer cooling capabilities considering the peak power provision of the whole infrastructure. Therefore, it is possible to build an exascale computer today, but if the power consumption trend of current HPC systems is followed, it would need a nuclear reactor to be powered [81]. Equally important, low-power devices such as embedded systems, either deployed using single-core or multi-core platforms, are being driven by applications such as video streaming and sensor data processing, showing an increase in peak power dissipation. Thus, some modern low-power microprocessors may offer operating modes consuming low power, but the more and more computationally intensive tasks require them to significantly increase the power consumption [10]. As a result, it can also be seen in Fig. 2.4*b*, that the total power consumption for HPC, PC and portable applications has been increasing over the years.

The power consumption has become a critical problem for increasing a microprocessor's performance because such a power is directly proportional to the number of transistors, the supply voltage and the frequency of operation of the chip. During the previous decades, power and energy management has become a prime design and operation dimension for many real-time platforms. In CMOS technology, which is still the dominant approach in the VLSI circuit design, the power consumption has both dynamic and static components, which are due to the system activity and leakage dissipation, respectively. It can be seen that the number of transistors being integrated into a single chip approximately doubles every two years to keep providing desirable processor performance (according to the so called Moors's law). But to keep increasing the transistor density it is necessary to reduce the size of each transistor, which makes quantum effects of the material less neglectable, and also makes it necessary to reduce the supply voltage and frequency of operation, thereby reducing the dynamic power consumption of the chip. Even though the threshold voltage has also been lowered, the gap between supply and

threshold voltages has been reduced, which leads to a significant increase in the leakage consumption, because the smaller the gap, the higher the subthreshold dissipation. Consequently, the static power consumption has become as important as the dynamic power consumption. Since the transistor density has been increasing over the years, and it is expected to continue growing , the static power of each integrated circuit and the total power density have been increased too, generating a high power dissipation and energy consumption in computing platforms. High power density leads computing system designers to deal in many cases with excessive thermal dissipation in one or multiple points of the platform [88].

Multiple power models have been proposed in the literature to predict dynamic power consumption in a processing unit. The most common model established in [88] states that the dynamic power is proportional to the switching capacitance C, the supply voltage value V_{dd}^{CPU} and the frequency of operation f^{CPU}, as shown in Eq. 2.3.

$$P_{dyn} = C \cdot (V_{dd}^{CPU})^2 \cdot f^{CPU} \tag{2.3}$$

The leakage power, also known as static power, is generated by leakage currents passing through active transistors. The values of such leakage currents are dependent on the temperature of the silicon and processor voltage. A high power consumption tends to generate a high temperature, which in turn aggravates the power consumption of the system. The leakage power is formulated as:

$$P_{leak} = N_{gate} \cdot (V_{dd}^{CPU}) \cdot I_0 \cdot \left[AT^2 \cdot e^{\frac{\alpha V + \beta}{T}} + B \cdot e^{\gamma \cdot V + \sigma} \right], \tag{2.4}$$

where T and (V_{dd}^{CPU}) are the temperature and supply voltage, respectively. N_{gate} is the number of gates in the circuit, and T_0 is the reference leakage current. A, B, α, β and γ are technology dependent constants [89].

In this dissertation we assume that the CPU working frequency is linearly proportional to the CPU supply voltage V_{dd}^{CPU}, and the dynamic power consumption can be formulated

as $P_{dyn} = C2 \cdot (V_{dd}^{CPU})^3$, where $C2$ is an architecture-dependent constant. On the other hand, the static power has a very complicated relationship between leakage current and temperature, as seen in Eq. 2.4, but it can be approximated to a linear function as follows: $P_{leak} = C0 + C1 \cdot \mathrm{T}_j(t)$, where $C1$ and $C2$ are also architecture-dependent constants and $\mathrm{T}_j(t)$ is the CPU junction temperature [28]. Thus, the total CPU power can be formulated as: $P^{CPU} = C0 + C1 \cdot \mathrm{T}_j(t) + C2 \cdot (V_{dd}^{CPU})^3$. Further, we assume that each task τ_i may have a different switching activity, modeled using μ_i (with $\mu_i \in (0,1]$), defining how intensively the CPU is consuming dynamic power by the execution of τ_i. Hence, the CPU power associated with each task τ_i can be reformulated as follows:

$$P_i^{CPU} = C0 + C1 \cdot \mathrm{T}_j(t) + \mu_i \cdot C2 \cdot (V_{dd}^{CPU})^3 \tag{2.5}$$

As seen in chapter 1 an increment in power consumption often increments the temperature of operation of many single-core and multi-core platforms in different applications, such as embedded systems and HPC systems. Multiple power models have been proposed in the literature to predict the thermal behavior of an electronic circuit, but the majority of them are based on the analogy between the flow of current in an electronic circuit and the flow of heat in any thermal device.

Thus, in this dissertation, similarly to multiple research works [28,90], we use a model elaborated based on the commonly known lumped RC thermal model [91] (as shown in Fig. 2.6). Specifically, assuming a fixed ambient temperature (T_{amb}), let $T(t)$ denote the temperature at time t. Then, we have:

$$RC\frac{dT(t)}{dt} + T(t) - RP(t) = T_{amb} \tag{2.6}$$

where $P(t)$ denotes the power consumption in Watts at time t, and R and C denote the thermal resistance (in $J/^{\circ}C$) and thermal capacitance (in $Watts/^{\circ}C$), respectively.

Researchers in both academia and industry have proposed numerous techniques to manage the power consumption and minimize energy consumption in computing systems.

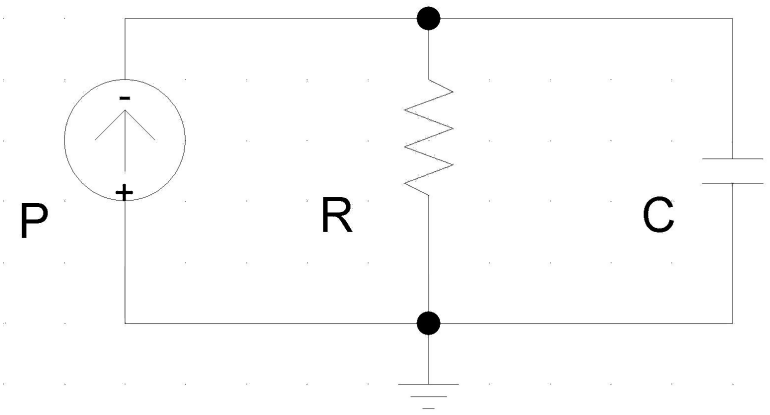


Figure 2.6: Lumped RC circuit example

Specifically for real-time embedded systems, two widely used techniques for reducing energy consumption in the processing unit are Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Power Management (DPM). DVFS approaches trade energy with performance by decreasing the voltage and the frequency of the processor to reduce the overall energy consumption. Since reducing the frequency increases the task execution times, a common objective in real-time systems is to derive processor/task speed values that still guarantee the timing constraints while minimizing the total energy. On the other hand, DPM techniques switch the processor to a low-power inactive state as long as possible, while guaranteeing that all real-time tasks will finish within their deadlines [44].

Some of those research works have proposed mechanisms to minimize the energy consumption while guaranteeing the timing constraints of real-time systems using a DVFS technique [92–97], or a DPM technique [98–100]. For instance, Awan et al. [96] address the problem of task-to-core allocation onto heterogeneous multi-core platforms such that the overall energy consumption of the system is minimized, considering both dynamic and leakage energy consumption. Using a DVFS technique, their approach considers core frequency set-points, tasks energy consumption and sleep states of the cores to reduce the energy consumption of the system. Additionally, Huang et al. [99] propose

online algorithms that adaptively control the power mode of a system, procrastinating the processing of arrived events as late as possible, providing solutions for preemptive EDF and fixed-priority scheduling policies. Bhatti et al. [98] propose a generic power/energy management scheme that takes a set of existing DPM and DVFS policies, each of which performs well for a set of conditions, and adapts at runtime to the best-performing policy for any given workload. Han et al. [101] study the problem of energy minimization for scheduling periodic fixed-priority tasks in multi-core platforms with fault tolerance requirements.

Similarly, multiple research works have noticed the importance of considering the increment of power and energy consumption, as well as their effects on the thermal management for real-time computing systems, as a very important optimization criteria for scheduling real-time tasks. The increment in power and energy consumption, as well as the increase in thermal emergencies in multiple computing platforms, are affecting the design of real-time systems too. Real-time schedulers must make decisions of execution of tasks not only considering their timing parameters, but also considering their power consumption and their potential effects over the system's operating temperature, in order to keep the system's power and temperature below manageable thresholds. Consequently, multiple scheduling techniques have been proposed to perform power and energy management, as well as thermal management for real-time systems [28, 90, 93, 102].

Some research works also have acknowledged the problem of manage the thermal operation of the processing unit as well, either in single-core or multi-core platforms, optimizing different parameters of the system, such as performance, energy consumption or reliability [45, 103–105]. For instance, Huang et al. [106] study the problem of how to maximize the throughput for a periodic real-time system under the given peak temperature constraint, when deployed on a single-core platform. Authors assumed that different tasks in the system may have different power and thermal characteristics, allow-

ing their proposed DPM scheduler to alternate the processor active/sleep modes. Their approach equally divide different tasks into m equal time sections in order schedule the execution of each section, lowering the system's peak temperature. Also, Fan et al. [28] present a closed-form analytical solution to calculate the system thermal steady-state energy consumption for a periodic voltage schedule on a multi-core platform, with the leakage/temperature dependency taken into consideration. Using these authors' approach, it is possible to quickly obtain the temperature dynamics in the system thermal steady state. Then, based on their temperature calculation method, they develop a closed-form solution of energy calculation for any scheduling period, particularly in system thermal steady state. Also, Han et al. [102] study the problem of how to determine if a periodic DVFS schedule for a multi-core platform is thermally feasible in satisfying a given peak temperature constraint. Ahmed et al. [46] propose a methodology for minimizing the peak temperature in embedded systems using periodic resource allocation. Egilmez et al. [107] develop a method to control the skin temperature in smartphones, by predicting the surface temperature and scaling the frequency of application processor by taking user comfort limit into consideration. Sha et al. [108] study the problem of how to maximize the computing performance of multi-core platforms without violating their peak temperature constraint and present a novel technique to maximize the throughput of the platform. Hettiarachchi et al. [109] propose a methodology to design predictable real-time systems in an unpredictable thermal environment where the environmental temperature may dynamically change, by allowing the system to adjust the scheduling according to a predefined set of performance modes.

Additionally, some research works have focused on solutions to the thermal management problem on HPC platforms. For instance, in [110] authors study the thermal-aware allocation of virtual machines in data centers, in [111] authors characterizes the thermal behavior of HPC systems using machine learning methods in order to enhance the sched-

uler by reducing the number of hot-spots in the system, and in [85] authors present a survey of thermal-aware scheduling techniques for green data centers.

2.3 Memory-Aware Scheduling

In this section, we present some preliminaries and review existing works related to scheduling mechanisms considering memory characteristics and issues. First, we review preliminary concepts and existing works related to mechanisms aimed to improve the predictability of the system, considering the timing delays imposed by task preemption scheduling methods and shared cache architectures. Second, we review preliminary concepts and existing works related to mechanisms enhancing the performance and predictability, by considering power and thermal issues.

2.3.1 Shared Cache Memory

Modern computing systems supplement the main memory of the computer with a set of local registers and one or more levels of cache in a memory hierarchy, where the cache level closest to the processing core(s) is small and fast, but contains only a subset of the contents of memory. Caches are transparent to the programmer, with hardware determining which parts of memory should be placed at any given time in the cache. Cache memory is the part of the memory hierarchy that has seen the most change over the years, increasing the size and number of cache layers that has been introduced between the register file and memory [31]. For the sake of scalability, flexibility, and to deal with power limitations, it has become mainstream to group multiple cores sharing a local cache memory.

The majority of the algorithms proposed to guarantee the schedulability of real-time systems (e.g. RMS or EDF), especially hard real-time ones, assume that tasks are fully

preemptive. This means that tasks can be suspended at arbitrary points in favor of higher-priority tasks. Preemption simplifies the schedulability analysis but introduces a runtime overhead (sometime called preemption cost) during task execution, due to the context switching between tasks, the pipeline invalidation delay, and the cache-related preemption delay (CRPD), especially for shared caches in multi-core platforms [16, 17]. The preemption cost is often assumed to be constant and speed independent. However, the CRPD is defined as the delay that a preempted job incurs due to a loss of cache affinity after resuming execution. This delay may vary greatly, depending on the architecture characteristics and the workload characteristics. Thus, such CPRD delay introduces a significant source of uncertainty in the timing analysis of real-time systems. In essence, the sharing of local cache memory helps to improve the average case execution time of each task, but can be hazardous to the estimation of the worst-case execution time (WCET). This is because the number of memory accesses, locations in time, and bus loads originated from other concurrent tasks are difficult to determine precisely. Fig. 2.7 shows an example of cache-related preemption delay. It can be seen that a low-priority task τ_1 with $T_1 = 10ms$ and $C_1 = 5ms$ (Fig. 2.7a) is scheduled along with a high-priority task τ_2 with $T_1 = 10ms$ and $C_1 = 3ms$ (Fig. 2.7b). However, if τ_2 unloads sufficient cached data of τ_1, C_1 increases and the second job of τ_2 misses its deadline (Fig. 2.7c) [112].

Multiple research works have acknowledged the problem of preemption cost, and specifically the CRPD increasing the uncertainties in the execution of tasks, which leads to even more pessimistic estimated WCET for each task, and one of the most common methodologies to tackle such a problem is based on the concept of *cache partitioning*. Shared cache partitioning approaches are widely applied in both general-purpose and real-time computing systems [113–116]. However, since a major source of pessimism in WCET estimation comes from shared cache memories, cache memory partitioning has proven to be one of the most effective methods to improve the predictability and

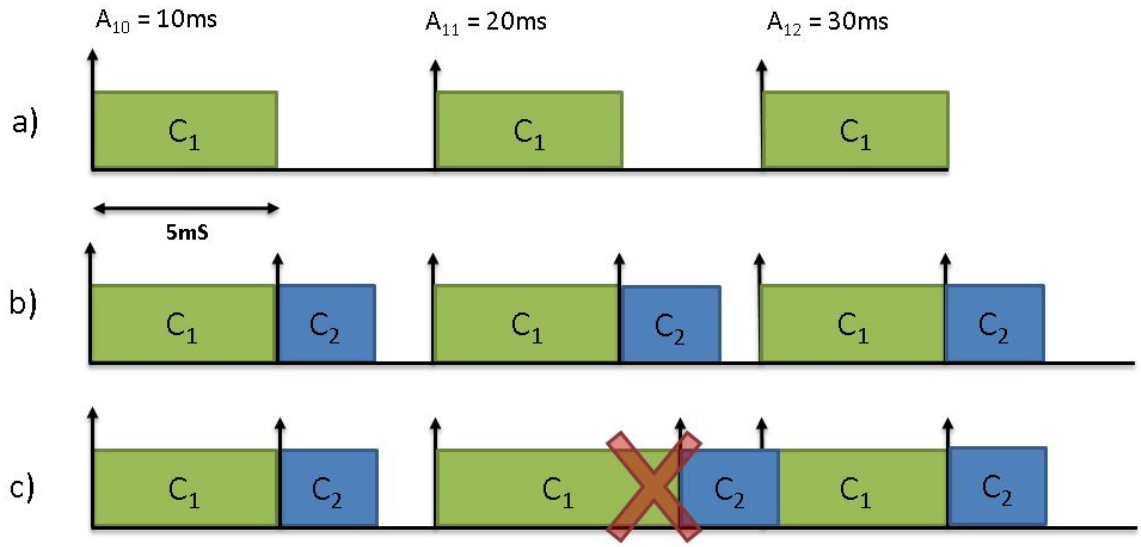


Figure 2.7: Example of cache related preemption delay. A low priority task τ_1 with $T_1 = 10ms$ and $C_1 = 5ms$ (a), is scheduled along with a high priority task τ_2 with $T_1 = 10ms$ and $C_1 = 3ms$ (b). If τ_2 unloads sufficient cached data of τ_1, C_1 increases and the second job of τ_2 misses its deadline (c) [112].

schedulability of real-time systems. Fig 2.8 shows a typical configuration of computing platforms with two cores and two levels of cache memory. The first level is private to each core, but the second level is shared among the two cores. Such second level of cache memory is divided into partitions. In the figure, two partitions are assigned privately to core 0 and one partition is assigned privately to core 1. Cache partitions can be managed at the hardware level, the compiler level or the operating system level. Therefore, it is possible to assign partitions privately to, not only cores, but also to real-time tasks. In essence, WCETs are bounded and controlled much more tightly when the cache is partitioned. This allows the estimation of real-time tasks' WCETs relatively tight, yet safe, which promotes processor utilization in both single-core and multi-core platforms.

The main goal of implementing shared cache partitioned approaches, in hard and soft real-time embedded systems, is to improve the predictability of the WCET, and consequently improve the response time of the real-time system [62, 117–121]. For example, authors in [122] propose a compiler-based method that partitions the cache among

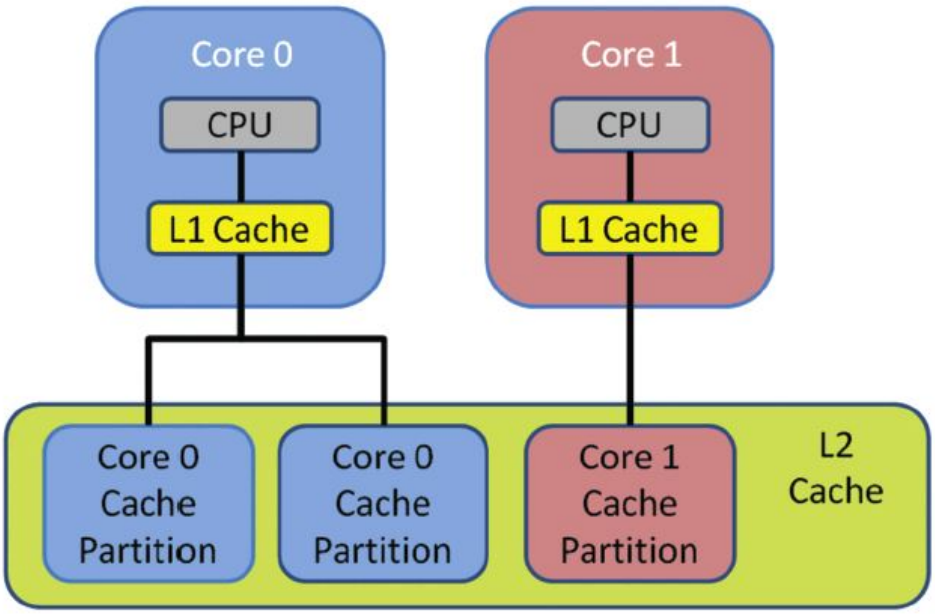


Figure 2.8: Example of dual-core configuration with cache partitioning [12]

tasks at compilation time. Authors in [113] propose a dynamic cache partitioning scheme that decreases the allocated spaces to the faster threads and allocates them to the critical ones to minimize the system overall response time. Authors in [114] and [115] suggest a software-based page coloring technique, while in [116] a hardware-based partitioned method is introduced.

Existing work on cache partitioning can be largely categorized into two groups: *cache allocation policies* and *cache management schemes* [123]. The first ones focus on policies to dictate how to allocate available cache resources to different tasks to achieve different objectives, such as fairness, priorities, and performance maximization (e.g. [116, 118, 124]). The second ones intend to enforce, by means of hardware or software, the distribution of the outcomes of the cache allocations so that each program can access its allocated cache memory (e.g. [62, 125, 126]). For instance, at the operating system level, cache coloring is an important cache allocation policy to optimize the performance of real-time and general purpose systems by cache allocating the contiguous pages from the cores

point of view. This technique sets up the virtual-to-physical-address translation so that no two tasks access the same cache set in the shared cache and hence one task cannot evict a cache block that another task has fetched to the shared cache [127].

As mentioned in chapter 1, multiple research works have realized the need to address the problems of the increasing power consumption in memory systems and thermal issues related to such a high power. Some of those research works have focused on the timing problems seen by real-time systems when power and thermal issues are taken into consideration. The power consumption of on-chip cache memories is usually assumed to be a constant value in many research works [128]. Since cache memories are implemented close to the CPU, their power can be associated with the power consumed by the entire CPU. Unlike cache memories, external main memory manufactured with DRAM technology usually has an additional power consumption, comparable to that of the CPU in today's platforms and dependent on multiple factors, e.g. the memory capacity and bandwidth. Thus, in what follows, we provide a general introduction of the role and organization of DRAM memories, along with the challenges of real-time system design, considering main memory power and thermal issues.

2.3.2 Main-Memory Power and Thermal

DRAM devices are designed utilizing very complex architectures that have evolved through time to deliver high-volume storage at low cost per bit, in the fastest way. A DRAM memory is divided into ranks, as shown in Fig. 2.9, and each rank is divided into multiple banks. As shown in Fig. 2.10, bank-level parallelism allows each bank in the memory to be accessed in parallel, so the memory space is interleaved among all the banks. Each bank comprises a *row-buffer* and an array of storage cells organized as *rows* and *columns*, as shown in Fig. 2.11. In order to access data, the memory controller (MC) activates a

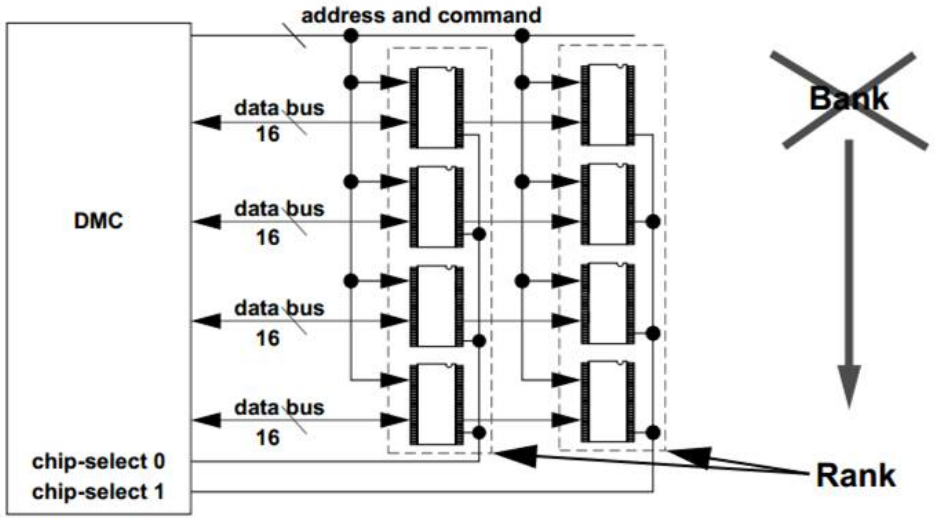


Figure 2.9: DRAM Rank organization [13]

row of a certain bank, copying all its columns on the row-buffer. An example of DRAM system organization is shown in Fig. 2.12. The access of data already on the row-buffer is faster than the data stored in the array of cells [13]. Since banks are interleaved, any core in the system can access any bank. If two applications running in parallel on different cores access two different rows in the same bank, they might force the MC to continuously pre-charge the row-buffer and open a new row every time an access is performed, making the latency of each memory request variable.

In this dissertation, we build our memory model similarly as numerous research works, based on the DRAM power model published by Micron [129] [130]. The DRAM power is directly proportional to the number of memory requests of each task τ_i (H_i maximum), and bounded by the DRAM chip maximum bandwidth BW_{max}^{DRAM} in MB/s. Therefore, each task τ_i has a single associated DRAM power value determined solely by its memory access rate.

The DRAM power consumption is composed of three components: background power, active power and read/write power. The *background power* varies with the CKE signal

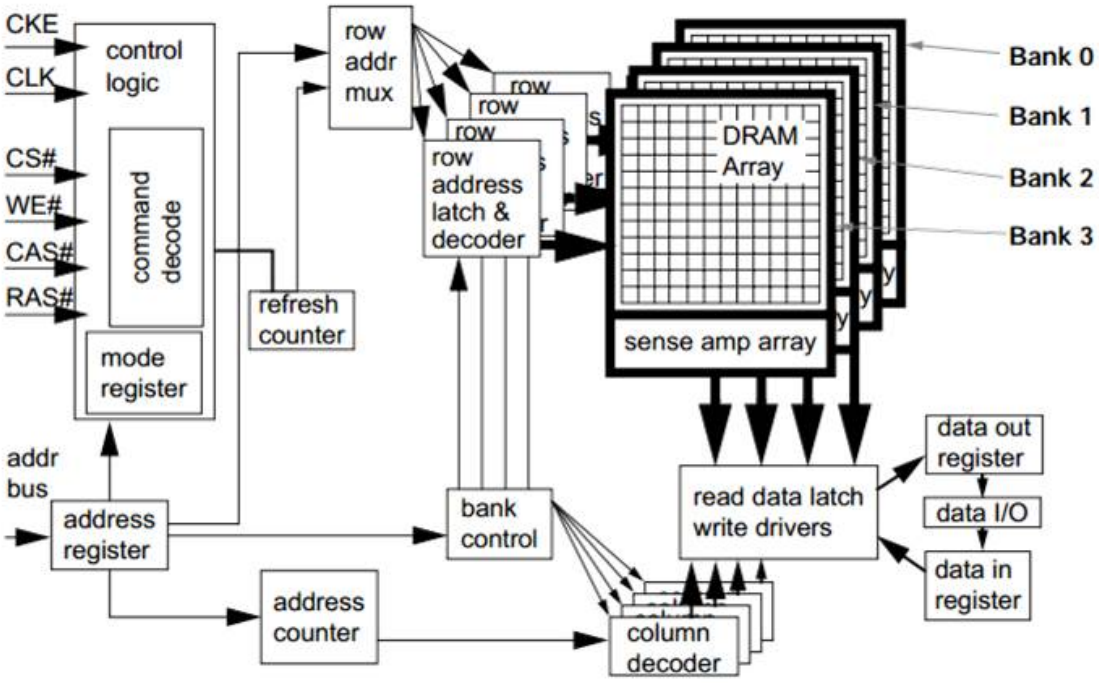


Figure 2.10: DRAM Bank organization [13]

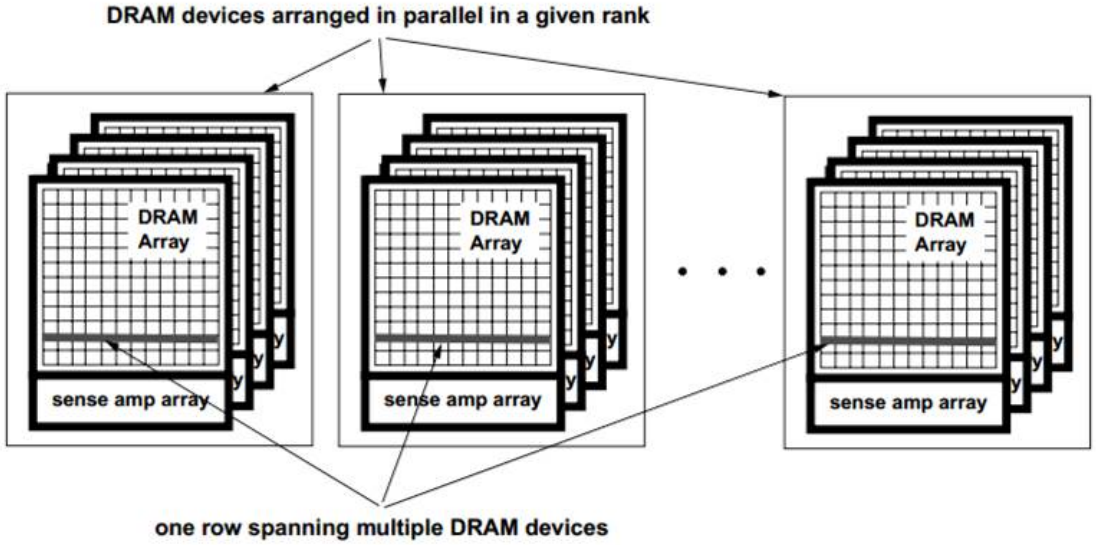


Figure 2.11: DRAM Row organization [13]

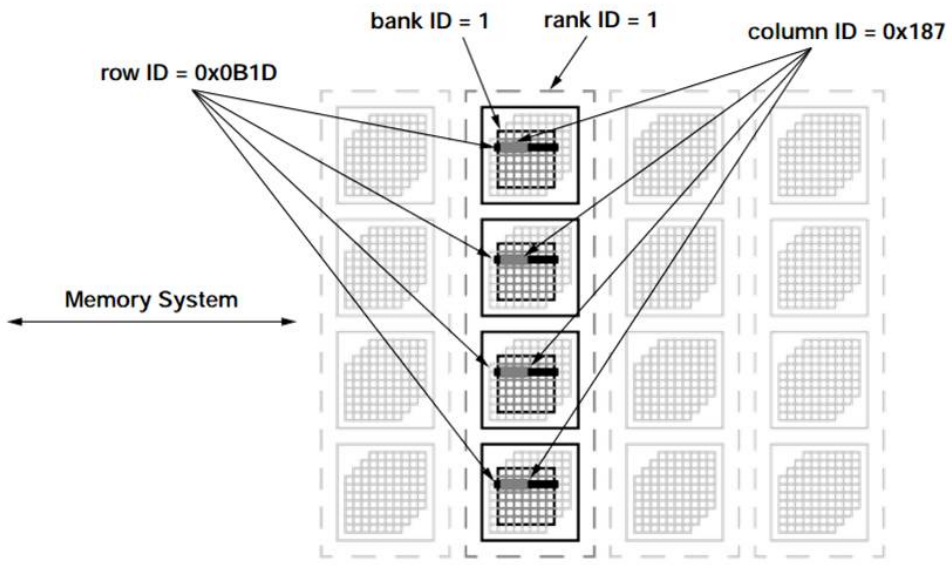


Figure 2.12: DRAM System organization [13]

of the DRAM chip. CKE is the master on-off switch. When the CKE signal is low, the DRAM goes off into a Power-Down state, consuming less power than active or stand-by. If the CKE signal is high the DRAM chip is able to receive commands from the MC. To ensure timely response, we assume that the DRAM does not enter the power-down mode during the execution of any task. The *background power* also contains the power for DRAM refreshes, i.e. P_{REF}. The *active power* depends on the currents to decode the command/addresses and transfer the data from the DRAM array to the sense amplifiers, and vice versa. The *read/write power* is related to the currents to place or store data to or from the bus. Consequently, the total power consumed by the DRAM when the CPU is executing a task τ_i is as follows:

$$P_i^{DRAM} = (P_{bg\_i} + P_{act\_i} + P_{RW\_i}) \cdot N_{Chip}, \tag{2.7}$$

where N_{Chip} is the total number of DRAM chips in the rank. Finally, for our experiments we consider the DDR3 manufacturing technology. Such DDR3 technology offers three

different power-down modes to save power. We are focused on the *self-refresh* power-down mode with power consumption denoted as P_{SR}^{DRAM} [130].

Main memory systems, especially systems built upon DRAM memory technology, introduce an additional level of uncertainty in the WCET estimation for real-time tasks. For instance, the bi-directional data path of the memory requires several cycles to switch from read to write and vice versa. Also, to prevent data loss, the memory must occasionally be refreshed before executing the next request and the added refresh time may be longer than the time to serve the request itself. These effects make the latency of each memory request variable. A critical problem seen in main memory systems is the so-called *bank-sharing problem*. This problem is similar to the sharing of cache spaces by multiple processing cores. When a task references a specific memory position, the row containing such a memory position is stored in a row-buffer, making successive references faster. However, if another task references a memory position in a different row, but in the same bank, the previous row has to be stored back and the new one is stored in the row-buffer. This represents a problem when real-time tasks are accessing the same bank address space, due to the unbounded delay added due to the interference. To avoid the bank-sharing problem, an approach called *bank partitioning* has been proposed, where each core is assigned with a set of exclusive banks. Some bank partitioning methods are software-based [127, 131, 132], and others are hardware-based [133, 134]. An additional mechanism proposed to guarantee the delay time each task sees from main memory consists of managing the bandwidth assigned to different real-time tasks [135, 136] by throttling the number of memory accesses of each core, in order to guarantee a specific bandwidth to critical tasks in the system. However, the implementation of techniques, such as bank partitioning or memory bandwidth throttling, brings and additional increment of the capacity and bandwidth that a memory system must comply with. This issue exacerbates the already high power consumption and excessive system temperature problem.

Multiple research works have proposed micro-architectural and system-level techniques for managing the power consumption, and also for managing the thermal behavior of the system [137–141]. Many of them propose the synergistic modification of operation modes and DVFS on CPU and memory devices [49]. Authors in [142] propose for embedded systems to consider the dynamic change of frequency of L2 cache memory, besides processors and main memory, in a multidimensional frequency scaling fashion, to improve the energy efficiency of the system. By evaluating different frequency levels, it is shown that the energy-delay-product (EDP) can be improved up to 46.4% when compared to the standard way that the frequencies are configured, without impacting performance. Other works, such as [143], adopt the concept of group of applications, which contains thread group and memory rank group. Their proposed mechanism manages a group of applications, by simultaneously scaling CPU frequency and controlling memory power mode to reduce both CPU and memory power.

Some works, such as [144], work with GPU systems. Authors in [144] seek to reduce power with minimal performance degradation in high-performance GPUs, by tuning the processing units' frequency, number of active computing units, and memory bandwidth. By tracking the time-dependent computing and memory demands for each task, the corresponding hardware power configurations of the core and memory system can be set to reduce overall platform power and thereby improve energy efficiency with minimal compromises in performance.

Authors in [145] present a model-based methodology that takes computation-specific properties into account, which guides power allocations for CPU and DRAM domains to maximize performance. Their methodology can predict the performance impacts of the power capping allocation schemes for different types of computations from real applications with absolute mean error of less than 6%. Also, authors in [146] present energy-management algorithms that coordinate core and DRAM frequency scaling under a spec-

ified energy budget. Additionally, authors in [147] propose a DRAM frequency selection method based on memory usage. The proposed method was implemented and tested with embedded Linux on a system equipped with a multi-core processor and 2GB LPDDR3 DRAM. Their method enhances energy efficiency of the device by up to 18%.

Regarding real-time systems, for instance, research works such as [148] propose a methodology to offer guarantees to legacy applications implemented on autonomous helicopter-style aircrafts. The method relies on NoC architectures and a DRAM controller contention-aware mechanism, and is based on the existing interference-sensitive WCET computation and the memory bandwidth throttling mechanism.

Authors in [149] investigate system-level thermal-aware data/task mapping policies for 3D memory architectures. Over a simulation framework, different workloads from a combination of the PARSEC benchmark suite are scheduled on a many-core 3D platform with a DRAM layer on top of a logic layer. Memory-bounded benchmarks from the PARSEC suite (such as canneal and streamcluster) have significant performance improvement due to an increase in the memory access bandwidth. However, 3D many-core systems consistently show higher peak temperatures [150].

During the past decade, 3D memory-processor integration has received considerable attention in the literature [151–153], and multiple research studies have been proposed to manage the thermal problems in 3D integration technology. Meng et al. [128] introduce a framework to model on-chip DRAM accesses and analyze performance, power, and temperature trade-offs of 3D systems. Their architecture focuses on one single layer of logic and one layer of DRAM memory. Chen et al. [154] characterize the thermal and performance behavior of the target architecture when the voltage and frequency levels of cores and DRAMs are synergistically controlled, targeting an architecture with multiple layers of DRAM memory. Some studies propose to manage the power and thermal parameters in 3D ICs by performing memory mapping techniques [155, 156]. Other thermal man-

agement approaches for 3D architectures, such as [157], propose to reduce temperature variance and the peak temperature of a 3D multi-core processor and stacked DRAM by thermally-aware thread migration among processor cores.

2.4 Summary

In this section, we presented the essential pertinent of our research and reviewed some closely related works in the literature. We first presented a general overview of the basic concepts and critical techniques in real-time scheduling. Particularly, we introduced different categorizations of real-time scheduling and two important scheduling methodologies for single-core scheduling (RMS and EDF). Next, we presented an overview of the periodic resource model and the concept of "resource server" used in scheduling methods to provide resource isolation to real-time tasks and allow the implementation of compositional real-time scheduling. Then, we presented some preliminaries for power and thermal-aware scheduling including the basic power and thermal models found in the literature. Also, we discussed some important research works on CPU-related power and thermal-aware scheduling. Finally, we presented some preliminaries of CPU and memory co-scheduling, including the DRAM memory technologies' power model, along with some discussion of related works, including works considering 3D integrated platforms.

In this dissertation, our goal is to develop effective scheduling mechanisms for hard real-time systems to guarantee timing constraints, while satisfying other constraints such as peak temperature of operation, when the scheduling of tasks considers the effects and delays imposed by memory devices. In the following chapters, i.e. chapters 3, 4 and 5, we present our contributions. Then, we conclude this dissertation in chapter 6.

CHAPTER 3

CACHE ALLOCATION FOR FIXED-PRIORITY REAL-TIME SCHEDULING ON MULTI-CORE PLATFORMS

As stated in previous chapters, in this dissertation we study the problem of how to allocate cache memory that is accessible by multiple processing cores when scheduling fixed-priority real-time tasks based on the rate monotonic scheduling (RMS) policy. Thus, we first present our research on cache allocation for real-time scheduling on multi-core platforms.

Since the WCET of a real-time task varies with its cache allocation, our research problem involves two intertwined problems: *i)* how to allocate the available cache memory partitions among all tasks, and *ii)* how to map each task to a core in the multi-core platform. One simple approach to partition the cache memory is to allocate the cache memory in such a way that it minimizes the *normalized resource usage* [158] —which includes both CPU utilization and memory utilization— for each task. However, the cache allocation that optimizes the resource usage for a single task does not necessarily optimize that for the entire task set. To map tasks to multiple cores and optimize CPU resource usage is a classical NP-hard problem.

It has been a well-known fact that harmonic tasks (tasks with periods being integer multiples of each other) can utilize CPU resource more effectively, i.e. with CPU utilization as high as 1 [159]. However, how to take the interplay of cache partitioning, execution time variations and task harmonic relationship into considerations to deal with cache allocation and task mapping in an integrated manner is the challenging problem we want to study in this chapter.

The rest of the chapter is organized as follows. Section 3.1 describes the most related research projects. Section 3.2 describes the architecture and real-time system models and shows a cache partitioning example. Section 3.3 and section 3.4, describe in detail our

first and second solution approach, respectively. Next, section 3.5 shows the results for the conducted experiments, and finally, we present the summary in section 3.6.

3.1 Related Work

The large inter-task interferences due to increased resource sharing (such as shared buses and memory) on multi-core platforms have severely undermined the predictability of real-time systems [39, 40]. For the sake of scalability, flexibility, and to deal with power limitation in the era of "dark silicon," it has become mainstream to group multiple cores sharing a local cache memory [160] [161] [162].

The sharing of local cache memory helps to improve the average case execution time of each task, but can be hazardous to the estimation of the worst-case execution time (WCET). One major problem in estimating the WCET bounds on multi-core systems is the unpredictability of the workload on other cores. Therefore, the number of memory accesses, locations in time, and bus loads originated from other concurrent tasks are difficult to determine precisely [40]. To assume the worst case scenario for each factor can be extremely pessimistic and nullifies the extra computational capacity of the multi-core platforms in the design of real-time systems.

In a first stage in this dissertation, we study the problem of how to allocate the cache memory that is accessible by multiple processing cores when scheduling fixed-priority real-time tasks based on the rate monotonic scheduling (RMS) policy. Thus, with more isolated memory accesses, each real-time task can avoid or reduce considerably the inter-task memory interferences. Therefore the WCET can be more accurately bounded and CPU utilization can be significantly increased. It is noteworthy to mention that the fixed-priority multi-core partitioned scheduling scheme is one of the most commonly used scheduling mechanisms for real-time system design [23], due to its advantage of better predictability. Besides, it is supported by almost all real-time operating systems available

on the market due to its low overhead and simplicity in implementation, and it is still the method of choice in industry. We assume that each real-time task will be executed on a dedicated processing core, and its WCET, for a specified cache size, can be estimated beforehand using strategies such as those presented in [163].

Since a major source of pessimism in WCET estimation comes from shared cache memories, cache memory partitioning has proven to be one of the most effective methods for managing the shared fast local memory while optimizing other design objectives, such as performance maximization [164], quality-of-service (QoS) enhancement [165], and fairness [114], and also to improve the predictability and schedulability of real-time systems [62, 114, 118, 119, 126, 127]. This method partitions cache memory among programs and cores to reduce cache contention between tasks and/or cores. By isolating real-time task memory accesses, cache memory partitioning can avoid or considerably reduce the inter-task interferences, and therefore reduce the uncertainty when bounding the WCET and improve the core utilization.

Existing work on cache partitioning can be largely categorized into two groups [123]: *cache allocation policies* and *cache management schemes*. The first ones focus on policies to dictate how to allocate available cache resources to different tasks to achieve different objectives, such as fairness, priorities, and performance maximization (e.g. [116, 118, 166]). The second ones intend to enforce, by means of hardware or software, the distribution of the outcomes of the cache allocations so that each program can access its allocated cache memory (e.g. [62, 125, 126, 167]).

From this perspective, we are interested in developing static cache allocation policies for real-time systems to enhance the predicability and schedulability when scheduled in a multi-core environment. Unlike our proposed allocation policies, some techniques have been proposed for single-core platforms [112, 168], and some others use a non-preemptive EDF policy for intra-core scheduling [169, 170]. A few approaches that have been pub-

lished are closely related to our work. Chang et. al. [2, 158] develop a series of algorithms for real-time systems scheduled based on EDF in island-based multi-core real-time systems with local and global heterogeneous memories. The algorithm, so called Island Based Real-Time Scheduling for Multi-Core Islands (IBRT-MCI), intends to optimize the system resource (CPU and fast local memory) usage for a single task. A variant of this algorithm is also introduced in a later publication [2], in which the intra-core scheduling is performed according to RMS, i.e. IBRT-MCI-RMS. As discussed later in this chapter, the optimal solution that can optimize the system resource usage for a single task does not necessarily optimize that for the entire task set. Also, as we show in our simulation results, incorporating period relation into cache allocation and task mapping can significantly improve the schedulability of real-time systems. Kim et. al. [171] propose a cache allocation policy that relies on page coloring as the cache management scheme. Different from our approach, their algorithm assigns cache units privately to cores instead of tasks, thus allowing intra-core cache units sharing. This alleviates the memory co-partitioning problem due to the page coloring management scheme, but increases the predictability analysis complexity. Suzuki et. al. [127] propose two algorithms as cache allocation policies, taking into consideration the cache memory partitions and the main memory banks assigned to each task. Unlike our approaches, such algorithms assume EDF as intra-core scheduling policy instead of RMS.

Alternatively, Fan and Quan [1] present a new multi-core partitioned scheduling algorithm, the Harmonic-Fit Partitioned Scheduling (HFPS) algorithm, for fixed-priority sporadic task systems. The authors exploited the fact that harmonic tasks or tasks close to harmonic can utilize the processor more efficiently increasing feasibility. Particularly, HFPS allocates tasks group by group in order to find the best combination in terms of system utilization, maximizing the mutual harmonicity among the task scheduled on a particular core. Their experimental results show that their proposed algorithm can signif-

icantly improve the scheduling performance compared with previous scheduling parameters, such as RM-Next-fit, RM-First-fit and RM-Best-Fit. However this work assumes a WCET for each real-time task as a single and predetermine value, non-dependant of the amount of fast local memory assigned to the task. Even for tasks that are not entirely harmonic, Fan and Quan [172] show that mapping tasks closer to harmonic together into one processing core can greatly improve the feasibility of real-time systems. The problem is how to take into consideration both factors, i.e. the harmonic relationship and variable execution times, to allocate cache memory and map tasks to cores.

We propose two algorithms in this chapter. The first algorithm combines two existing works: one based on fast local memory partitioning [158], and the other one, on harmonic-based scheduling [172]. The second algorithm is a more elaborated approach that can judiciously choose the cache size for each task and also exploit task harmonic relationships. Therefore, it can significantly improve the system resource usage and task set schedulability. We use a third party data report of the cache performance for the SPEC CPU2000 benchmarks suite [173] to validate our approaches. The results show that our approach can significantly improve the schedulability of real-time tasks, i.e. up to four times, when compared with other scheduling mechanisms.

3.2 Preliminary

In this section, we introduce the architecture and the real-time system model used in this chapter. We also show an example to motivate our research.

Table 3.1: Example of Task Set and the WCET values for different m_i

	m_i																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	T_i
$C_1^{m_1}$	5	5	4	4	3	3	3	3	3	3	3	3	3	3	3	3	10
$C_2^{m_2}$	20	18	10	6	6	6	6	2	1	1	1	1	1	1	1	1	25
$C_3^{m_3}$	10	8	6	6	6	5	5	5	4	4	4	4	4	4	4	4	13
$C_4^{m_4}$	10	9	8	7	6	5	5	5	5	5	5	5	5	5	5	5	25

3.2.1 Architecture and System Model

The multi-core platform consists of a set of P homogeneous processing cores, denoted as P_k with $k = 1, 2, ..., P$. The cache memory is divided into a finite number of allocation units of the same size called *cache units*. The total number of cache units is denoted as B.

The task set consists of N independent implicit-deadline periodic tasks, denoted as $\Gamma = \{\tau_1, \tau_2, ..., \tau_N\}$, scheduled according to RMS. Each task τ_i, where $1 \leq i \leq N$, is characterized by its minimum inter-arrival time T_i. A finite number, denoted as m_i, of cache units are assigned privately to a single task τ_i executed by a core of the system, and its WCET varies with m_i, which is denoted as $C_i^{m_i}$. Therefore, the task set Γ may be characterized by a matrix like the one shown Table 3.1. In this table, the rows indicate each task belonging to the task set (four tasks for the example), and each column (except for the last one) indicates the number of assigned cache units m_i to each task ($1 \leq m_i \leq 16$). The numbers shown in the matrix correspond to each $C_i^{m_i}$ of each task. The last column indicates the period of each task.

Each task $\tau_i \in \Gamma$ is characterized by a CPU-utilization and a memory-utilization. We define $U_i^{m_i}$ as the CPU-utilization of τ_i, where $U_i^{m_i} = C_i^{m_i}/T_i$ and B_i as the Memory-utilization of τ_i where $B_i = m_i/B$. In the same way, we also define the CPU-utilization of a task set Γ as $U(\Gamma) = \sum_{\tau_i \in \Gamma} U_i^{m_i}$, and the total number of cache units used by a task set $q(\Gamma) = \sum_{\tau_i \in \Gamma} m_i$. When task $\tau_j \in \Gamma$ is assigned with a specific value of m_j, we reference its CPU utilization as U_j.

3.2.2 Cache Allocation Example

Before we present our algorithms, we first show an example, i.e. Example 3.1, of a cache allocation problem along with two possible solution methods: the first one, using a previous proposed technique, and the second one, using a simple inspection.

Example 3.1. *Consider a task set consisting of four tasks, as shown in Table 3.1, to be scheduled in a platform consisting of two cores, sharing a cache memory with 16 cache units.*

The problem defined in Example 3.1 has proven to be NP-hard. One solution for this problem, i.e. IBRT-MCI-RMS presented by Chang et al. [2], is to first allocate the cache space that can optimize the resource usage for a single task, and then transform this problem to the traditional bin-packing problem. To this end, they first define a metric, called *normalized resource usage*, to balance CPU and cache resource usage, as shown in the following definition.

Definition 3.1. *The* minimum normalized resource usage *[2] of task* $\tau_i \in \Gamma$*, denoted as* λ_i*, is defined as:*

$$\lambda_i = \min_{0 \leq m_i \leq B} \left(\frac{U_i^{m_i}}{P} + \frac{m_i}{B} \right) \tag{3.1}$$

Essentially, the minimum normalized resource usage of a task is the minimum sum of its normalized processor utilization and the normalized cache allocation. With task execution times and cache allocations given, the minimum normalized resource usage of a task can be readily identified. Table 3.2 shows the cache allocation results based on this approach. Columns m_i and C_i, and thus U_i are obtained based on Def. 3.1. Then, IBRT-MCI-RMS sorts tasks in a non-decreasing order with respect to their values of m_i, and packs tasks to cores with utilization bounded by the traditional Liu&Layland upper-bound [25]. In the case of a task set of two tasks, such bound is of 0.83. For this example,

Table 3.2: Motivation Example Solution Using IBRT-MCI-RMS [2]

τ_i	T_i	C_i	m_i	U_i	Sub Task Set Utilization	Sched. Cond.	Feasibility
2	25	6	4	0.24			
3	13	6	3	0.46	$U(\Gamma_{2,3}) = 0.70$	$U(\Gamma_{2,3}) \overset{?}{\leq} 0.83$	Core 1 - YES
1	10	5	1	0.50			
4	25	10	1	0.40	$U(\Gamma_{1,4}) = 0.90$	$U(\Gamma_{1,4}) \overset{?}{\leq} 0.83$	Core 2 - NO

as shown in Table 3.2, the total utilization for the subtask set with τ_2 and τ_3 (a value of 0.70) is less than the upper bound. However, the value of total utilization for the subtask set τ_1 and τ_4 (a value of 0.90) is larger than the utilization bound. Therefore, IBRT-MCI-RMS fails to schedule the task set of Example 3.1.

For the problem defined above, a feasible solution does exist. As shown in Table 3.3, by assigning 3 cache units to τ_1 and 2 cache units to τ_4, τ_1 and τ_4 would decrease their WCETs from 5 to 4 and from 10 to 9, respectively, making the task set comply with the schedulability condition defined by the Liu&Layland upper-bound. Besides, the total number of cache units used by the task set would be increased from 9 to 12, which is still less than 16. The numbers underlined in Table 3.3 represent the changed values from the solution shown in Table 3.2. This shows that, even though IBRT-MCI-RMS allocates cache space to optimize the resource usage (according to λ_i of Def. 3.1) of a single task, the local optimum solution cannot guarantee that the solution is globally optimal. In addition, it is well-known that period relationship of real-time tasks has a significant impact on their schedulability on a processor [75, 159]. The question is how to take it into consideration in cache allocation and task mapping to improve system resource usage and schedulability of real-time task sets.

Table 3.3: Motivation Example Solution by Inspection

τ_i	T_i	C_i	m_i	U_i	Sub Task Set Utilization	Sched. Cond.	Feasibility
2	25	6	4	0.24			
3	13	6	3	0.46	$U(\Gamma_{2,3}) = 0.70$	$U(\Gamma_{2,3}) \overset{?}{\leq} 0.83$	Core 1 - YES
1	10	4	3	0.40			
4	25	9	2	0.36	$U(\Gamma_{1,4}) = 0.76$	$U(\Gamma_{1,4}) \overset{?}{\leq} 0.83$	Core 2 - YES

3.3 Simple Harmonic-Based Cache Allocation Approach (HBCA1)

One way to exploit the period relationship among tasks is to simply incorporate the task period into the task mapping phase only. During the cache allocation phase, we can search a local optimal value for the parameters C_i and m_i for each $\tau_i \in \Gamma$ based on the metric λ_i described in Def. 3.1 [2]. Note that, after λ_i is defined, the WCET for each task is also defined. Then, we can employ the harmonic-based task mapping method (such as the one in [172]) to map tasks to multi-core platforms. We call this approach HBCA1 (Harmonic-Based Cache Allocation 1), which is shown in Alg. 1. In Alg. 1, we assume that all processing cores share the same cache memory. The algorithm can be easily extended to deal with the scenario of when processing cores share multiple cache memories.

While a harmonic task set can be schedulable with total utilization reaching as high as 1, not all tasks are harmonic. Therefore, to better exploit the harmonic relationship among tasks, one critical question is how *harmonic* a task set is. To this end, Fan et al. [172] introduce the concept of *primary sub-harmonic task set* and, based on it, they develop the *harmonic index* to quantify the harmonicity.

Definition 3.2. *[172] Given a task set* $\Gamma = \{\tau_1, \tau_2, ..., \tau_N\}$ *where* $\tau_i = (C_i, T_i)$*, let* $\Gamma' = \{\tau'_1, \tau'_2, ..., \tau'_N\}$ *be a harmonic task set with* $\tau'_i = (C_i, T'_i)$ *and* $T'_i \leq T_i$*. Then,* Γ' *is called a* Primary Sub-Harmonic (PSH) task set *of* Γ *if there exists no harmonic task set* $\Gamma'' = \{\tau''_1, \tau''_2, ..., \tau''_N\}$*,* $\tau''_i = (C_i, T''_i)$ *and* $T''_i \leq T_i$*, such that for* $T'_i \leq T''_i$ *for all* $1 \leq i \leq N$*.*

Algorithm 1 Simple Harmonic-Based Cache Allocation Approach (HBCA1)

Input: Γ, P, B, WCET Task Matrix
Output: Cache Allocation && Task Partition Results

1: $rem\_cacheunits = B$ /*Remaining cache units in memory*/
2: $rem\_Cores = P$ /*Remaining idle cores sharing mem.*/
3: $\Gamma_{TS} = \emptyset$; $\mathcal{P} = \{P_1, P_2, ..., P\}$;
4: **for all** $\tau_i \in \Gamma$ **do**
5: Find m_i such that: $\left(\frac{U_i^{m_i}}{P} + \frac{m_i}{B}\right)$ is minimum;
6: $C_i = U_i^{m_i} \cdot T_i$;
7: **end for**
8: **while** $\Gamma \neq \emptyset$ && $|\mathcal{P}| \neq 0$ **do**
9: Sort τ_i increasing order with respect to T_i;
10: $n = |\Gamma|$; $U_{TS} = -\infty$; $B_{th} = \frac{rem\_cacheunits}{rem\_Cores}$;
11: **for** $i = 1$ to n **do**
12: Construct Γ' (PSH task set of Γ) using DCT [75] with τ_i as base;
13: Sort all $\tau_j \in \Gamma$ in increasing order with respect to $\Delta U_j = U'_j - U_j$;
14: Γ_{k_j} = pick up k_j tasks from Γ such that:
15: (1) $U(\Gamma'_{k_j}) \leq 1$;
16: (2) $U(\Gamma_{k_j})$ is maximized;
17: (3) $q(\Gamma_{k_j}) \leq B_{th}$;
18: **if** $\{U(\Gamma'_{k_j}) \leq 1\}$ **AND** $\{U(\Gamma_{k_j}) > U(\Gamma_{TS})\}$ **then**
19: $\Gamma_{TS} = \Gamma_{k_j}$;
20: **end if**
21: **end for**
22: Assign Γ_{TS} *to* $P_k \in \mathcal{P}$;
23: $\mathcal{P} = \mathcal{P} - P_k$;
24: $\Gamma = \Gamma - \Gamma_{TS}$;
25: Recalculate $rem\_cacheunits$ and rem_Cores;
26: **end while**
27: **if** $\Gamma \neq \emptyset$ **then**
28: Return: Γ is not schedulable;
29: **end if**

Definition 3.3. *[172] Given a task set* $\Gamma = \{\tau_1, \tau_2, ..., \tau_N\}$ *where* $\tau_i = (C_i, T_i)$*, let* $\mathcal{PSH}(\Gamma)$ *denote the set of all PSH task sets for* Γ*. The* harmonic index*, denoted as* $H(\Gamma)$*, is defined as:*

$$H(\Gamma) = \min_{\Gamma' \in \mathcal{PSH}(\Gamma)} (U(\Gamma') - U(\Gamma)), \tag{3.2}$$

where $U(\Gamma')$ *and* $U(\Gamma)$ *are the overall system utilizations for* Γ' *and* Γ*, respectively.*

The lower a task set's harmonic index is, the *closer* it is to one of its primary sub-harmonic task sets and therefore more harmonic. As discussed by Fan et al. [172], one approach to identify sub-harmonic task sets for a given task set is to employ the *DCT* algorithm [75]. In addition, the schedulability of a real-time task set can be predicted based on its sub-harmonic task sets, as stated in the following theorem:

Theorem 3.1. *[75] Let* Γ' *be a sub-harmonic task set of* Γ*. Then,* Γ *is feasible on a single processing unit under RMS, if* $U(\Gamma') \leq 1$*.*

Alg. 1 first determines the local optimal cache allocation based on the metric λ_i described in Def. 3.1 (lines 4 to 7). Then, it packs tasks that are most harmonic to the reference task (τ_i) in a sub task set Γ_{k_j} and maximizes the task utilization (line 16). To prevent "greedy" tasks from hoarding all the available memory cache units, we set a *cache units allocation threshold* (CUAT), i.e. B_{th}, requiring that the total cache units allocated to tasks on the same sub task set Γ_{k_j} should not exceed B_{th} (line 17). In our approach, we define B_{th} as the average available cache units for each core. This procedure is repeated by taking each $\tau_i \in \Gamma$ as the reference task (*for loop* line 11). The schedulable task set with the highest utilization, i.e. Γ_{TS}, is allocated to a processing core (line 22). The CUAT is recalculated and the procedure is repeated for the rest of the tasks and cores, until there are no more tasks left or no more cores are available in the system (*while loop* line 8).

As an example, Table 3.4 shows the solution to the problem described in Example 3.1 using HBCA1. The two sections in the table correspond to the procedures to find the sub

task sets for Core 1 and Core 2, respectively. Columns labeled as C_i, m_i, T_i corresponds to the WCETs, allocated cache units, and periods of tasks. Columns labeled as T_i', U_i' and U_i show periods and utilizations of tasks in the PSH task sets. Tasks in Table 3.4 are sorted based on ΔU_i. Columns of ΔT_i, ΔU_i are the period and utilization differences between a task with its corresponding task in the PSH task set. Columns of $Cum.\ U_i$ and $Cum.\ U_i'$ are the sums of the values for U_i and U_i' for when each task in the row is added. For example, in the first PSH task set, for τ_1 the $Cum.\ U_i' = 0.5$, for $\tau_1 + \tau_2$ the $Cum.\ U_i' = 0.8$, and for $\tau_1 + \tau_2 + \tau_4$ the $Cum.\ U_i' = 1.3$, which is larger than 1, indicating that only τ_1 and τ_2 can be scheduled together in one core (according to Theorem 3.1).

At the beginning, there are four tasks in the task set to be scheduled, and therefore the algorithm generates four different PSH task sets, as shown in the four rows of the first section in Table IV. The first one generated is the best candidate to be scheduled in core 1 since the feasible sub-task set (i.e. τ_1, τ_2) has the largest accumulated utilization (i.e. $U(\{\tau_1, \tau_2\}) = 0.74$) among the four. Hence, τ_1 and τ_2 are scheduled to core 1. The algorithm continues allocating the remaining tasks, repeating the process. In this case, the algorithm generates two different PSH task sets. The second row in the second section of Table IV shows that $U(\{\tau_3, \tau_4\}) = 0.86$ and $U'(\{\tau_3, \tau_4\}) = 0.88 \leq 1$. This ensures that τ_3 and τ_4 can be scheduled to core 2.

The complexity of Alg. 1 mainly comes from the loop from lines 10-15 with a complexity of $O(n^2 log\ n)$. Since the loop will be executed for P times, the overall complexity of Alg. 1 is $O(Pn^2 log\ n)$. While Alg. 1 can successfully schedule the task sets in Example 3.1, one big limitation of this approach is its local optimum cache allocation, i.e. optimum from each task's perspective. In what follows, we develop a more elaborate cache allocation and task scheduling approach that considers the task harmonic relationship.

Table 3.4: Solution to Example 3.1 using HBCA1

CHOOSE SUB TASK SET FOR CORE 1												
1	τ_i	T_i	C_i	m_i	U_i	T_i'	U_i'	ΔT_i	ΔU_i	$Cum.U_i'$	$Cum.U_i$	
	τ_1	10	5	1	0.50	10	0.50	0	**0**	0.5	0.50	✓
	τ_2	25	6	4	0.24	20	0.30	5	**0.06**	0.8	0.74	✓
	τ_4	25	10	1	0.40	20	0.50	5	**0.10**	1.3	1.14	
	τ_3	13	6	3	0.46	10	0.60	3	**0.14**	1.9	1.60	
2	τ_i	T_i	C_i	m_i	U_i	T_i'	U_i'	ΔT_i	ΔU_i	$Cum.U_i'$	$Cum.U_i$	
	τ_3	13	6	3	0.46	13	0.46	0	**0**	0.47	0.46	
	τ_2	25	6	4	0.24	13	0.46	12	**0.22**	0.92	0.70	
	τ_1	10	5	1	0.50	6.5	0.77	3.5	**0.27**	1.69	1.20	
	τ_4	25	10	1	0.40	13	0.77	12	**0.37**	2.46	1.60	
3	τ_i	T_i	C_i	m_i	U_i	T_i'	U_i'	ΔT_i	ΔU_i	$Cum.U_i'$	$Cum.U_i$	
	τ_2	25	6	4	0.24	25	0.24	0	**0**	0.24	0.24	
	τ_4	25	10	1	0.40	25	0.40	0	**0**	0.64	0.64	
	τ_3	13	6	3	0.46	12.5	0.48	0.50	**0.02**	1.12	1.10	
	τ_1	10	5	1	0.50	6.26	0.80	3.75	**0.30**	1.92	1.60	
4	τ_i	T_i	C_i	m_i	U_i	T_i'	U_i'	ΔT_i	ΔU_i	$Cum.U_i'$	$Cum.U_i$	
	τ_2	25	6	4	0.24	25	0.24	0	**0**	0.24	0.24	
	τ_4	25	10	1	0.40	25	0.40	0	**0**	0.64	0.64	
	τ_3	13	6	3	0.46	12.5	0.48	0.50	**0.02**	1.12	1.10	
	τ_1	10	5	1	0.50	6.25	0.80	3.75	**0.30**	1.92	1.60	
CHOOSE SUB TASK SET FOR CORE 2												
1	τ_i	T_i	C_i	m_i	U_i	T_i'	U_i'	ΔT_i	ΔU_i	$Cum.U_i'$	$Cum.U_i$	
	τ_3	13	6	3	0.46	13	0.46	0	**0**	0.46	0.46	
	τ_4	25	10	1	0.40	13	0.77	12	**0.37**	1.23	0.86	
2	τ_i	T_i	C_i	m_i	U_i	T_i'	U_i'	ΔT_i	ΔU_i	$Cum.U_i'$	$Cum.U_i$	
	τ_4	25	10	1	0.40	25	0.40	0	**0**	0.40	0.40	✓
	τ_3	13	6	3	0.46	12.5	0.48	0.5	**0.02**	0.88	0.86	✓

3.4 Enhanced Harmonic-Based Cache Allocation Approach (HBCA2)

In order to increase the schedulability of the system, we propose a second and more elaborate approach. The second approach is called the HBCA2 (Harmonic-Based Cache Allocation 2), and is shown in Alg. 2. It does not allocate cache memory based solely on the relation of WCET and number of cache units for each individual task. Instead, HBCA2 first groups tasks according to their harmonic relationship. Then, it allocates memory cache units to tasks in a way that can decrease the task set CPU utilization the most, when assigned with the same or less number of cache units possible.

The first problem for HBCA2 is to identify the candidate sub-task sets that may be assigned to a single core. Since the harmonic task sets can better utilize CPU resources, one intuitive approach is to employ the harmonic index as defined in Def. 3.3 and allocate tasks with a high harmonic index to the same core. However, since the cache allocations have not been determined, and thus the WCETs are not available, the harmonic index defined in Def. 3.3 does not apply. As a result, we use a different harmonic index ($H_t(\tau_i, \tau_j)$) to quantify, for a given task set, how harmonic a task is to a reference task.

Definition 3.4. *Let* $\Gamma'_j = \{\tau'_1, \tau'_2, ..., \tau'_N\}$ *be a PSH task set of a task set* $\Gamma = \{\tau_1, \tau_2, ..., \tau_N\}$ *with* $\tau'_j = \tau_j$. *The* harmonic index of task $\tau_i \in \Gamma$ *with respect to task* $\tau'_i \in \Gamma'_j$, *denoted as* $H_t(\tau_i, \tau_j)$, *is defined as:*

$$H_t(\tau_i, \tau_j) = \frac{T_i - T'_i}{T_i}. \tag{3.3}$$

Note that the harmonic index defined in Def. 3.4 is independent of its WCET or cache allocation. Therefore, we can construct the PHS task sets and order tasks based on the new harmonic index before the cache allocation is performed. The question becomes how to allocate cache units to the selected tasks with a high degree of harmonic relationship.

We develop an incremental approach for the cache allocation. Specifically, we first set the number of cache units to be 1 (i.e. $m_i = 1$) for each task, i.e. the most unbalanced resource allocation when the CPU utilization is maximized and the memory utilization is minimized for each task (line 4). Tasks with high harmonic index values are grouped into one sub-task set Γ'_i, until (*i*) no task can be added to the sub-task set while keeping the task set schedulable; (*ii*) the total cache units are no more than CUAT, i.e. B_{th}, as explained before (line 12).

Since the number of total cache units for the selected tasks is less than B_{th}, an opportunity is presented to allocate more cache units to the selected tasks, i.e. Γ'_i. As these selected tasks decrease their execution times with more cache units, more tasks can be

assigned in the processing core without compromising the schedulability (*while loop* line 13).

To this end, we design a new metric $CRRI(\tau_j)$ (*Combined Resources Ratio Index (CRRI)*) as follows:

Definition 3.5. *Let $C_i^{m_i}$ and $C_i^{m_i+x}$ be the WCETs with respect to the (privately assigned) shared cache size of m_i and $m_i + x$ cache units. The Combined Resources Ratio Index (CRRI) of τ_i, denoted as $CRRI(\tau_i, m_i, x)$, is defined as*

$$CRRI(\tau_i, m_i, x) = \frac{\Delta U_i}{\Delta B_i} \tag{3.4}$$

where $\Delta U_i = (C_i^{m_i} - C_i^{m_i+x})/T_i$ (the decrement in CPU utilization for τ_i) and $\Delta B_i = x/B$ (the increase in memory utilization for τ_i), B is the total number of cache units in a shared cache.

CRRI is essentially a benefit/cost index for cache allocation to a task. A higher CRRI value means that the decrement of WCET of τ_i is larger with a smaller number of extra cache units assigned to it. Thus, the higher the value for *CRRI*, the better the resource usage efficiency. One by one the next tasks in line (according to the harmonic index order) are assigned to Γ_i' (line 14), making the task set unschedulable. Therefore, the number of cache units for the task with the highest *CRRI* value, so called the *Guilty-Task(GT)*, is increased until the task set is schedulable again, i.e. $U(\Gamma_i') \leq 1$, or the number of total cache units assigned to the task set exceeds B_{th} (*while loop* line 15). This procedure is then repeated until the maximum number of cache units allowed for tasks on each core is reached (line 24). If the next task in line cannot be added to the existing task set, the original cache allocation for the existing task set is recovered (line 23).

The complexity of Alg. 2 mainly comes from the loop from lines 8 to 30. Assuming that in the worst case each core can accommodate n tasks, the complexity of the loop is $O(n^3)$ and the overall complexity of the algorithm is $O(Pn^3)$.

Algorithm 2 Enhanced Harmonic-Based Cache Allocation Approach (HBCA2)

Input: Γ, B, P, WCET Task Matrix
Output: Cache Allocation && Task Partition Results

$rem\_cacheunits = B$ /*Remaining cache units in memory*/
$rem\_Cores = P$ /*Remaining idle cores sharing mem.*/
$\Gamma_{TS} = \emptyset$; $\mathcal{P} = \{P_1, P_2, ..., P\}$;
for all $\tau_i \in \Gamma$ **do** $m_i = 1$; $C_i = C_i^1$; **end for**
while $\Gamma \neq \emptyset$ && $|\mathcal{P}| \neq 0$ **do**
    Sort $\tau_i \in \Gamma$ by the increasing order of T_i;
    $n = |\Gamma|$; $U_{TS} = -\infty$; $B_{th} = \frac{rem\_cacheunits}{rem\_Cores}$;
    for $i = 1$ to n **do**
        Construct Γ' (PSH task set of Γ) using *DCT* [75] with τ_i as base;
        Sort $\tau_j \in \Gamma$ by the increasing order of $H_t(\tau_j, \tau_i)$;
        $step = 1$; $\Gamma'_i = \emptyset$;
        Γ'_i = pick up the first j tasks listed from Γ such that:
    (1) $U(\Gamma'_i) \leq 1$; (2) $q(\Gamma'_i) \leq B_{th}$;
        while $j \leq n$ **do**
            $j = j + 1$; $\Gamma'_i = \Gamma'_i + \tau_j$;
            while $U(\Gamma'_i) > 1$ && $q(\Gamma'_i) < B_{th}$ **do**
                Find $\tau_{GT} \in \Gamma'_i$ s.t: $CRRI(\tau_{GT}, m_{GT}, step)$ is max.;
                if τ_{GT} is unique **then**
                    $m_{GT} = m_{GT}$ + $step$; $step = 1$; recalculate $q(\Gamma'_i)$ and $U(\Gamma'_i)$;
                else
                    $step = step + 1$;
                end if
            end while
            if $U(\Gamma'_i) > 1$ **then** $\Gamma'_i = \Gamma'_i - \tau_j$; **end if**
            if $q(\Gamma'_i) \geq B_{th}$ **then** break; **end if**
        end while
        if $U(\Gamma'_i) > U(\Gamma_{TS})$ **then**
            if $\{|\Gamma_i| > |\Gamma_{TS}|\}$ **OR** $\{|\Gamma_i| == |\Gamma_{TS}|$ **AND** $q(\Gamma'_i) \leq q(\Gamma_{TS})\}$ **then** $\Gamma_{TS} = \Gamma'_i$; **end if**
        end if
    end for
    Assign Γ_{TS} *to* $P_k \in \mathcal{P}$; $\mathcal{P} = \mathcal{P} - P_k$; $\Gamma = \Gamma - \Gamma_{TS}$;
    Recalculate $rem\_cacheunits$ and rem_Cores;
end while
if $\Gamma \neq \emptyset$ **then** Return: Γ is not schedulable; **end if**

Similar to Alg. 1, Alg. 2 constructs the sub-harmonic task set based on each task using the *DCT* algorithm. As the *DCT* algorithm generates one PSH task set when each τ_i is taken as the reference task, the algorithm comes up with n different sub-task sets to be allocated to a core. These task sets may have different performances in terms of system utilizations, task numbers, and total numbers of cache units, which conflict with each other. To explore all the Pareto optimal solutions may lead to an extremely large search space and is not realistic. In our approach, we adopt a simple metric as follows to choose the best sub-tasks to map to a core: The chosen task set is the one that has the maximum $U(\Gamma'_i)$ value with the highest total number of tasks $|\Gamma'_i|$. If the task numbers are the same, then the one with the smaller total number of used cache units $q(\Gamma'_i)$ wins (lines 26 to 28).

As an example, Table 3.5 shows the solution to the problem described in Example 3.1 using HBCA2. Data is presented in the same way as in Table 3.4, but tasks in Table 3.5 are sorted based on ΔT_i. Unlike HBCA1, algorithm HBCA2 is able to notice that by assigning three extra cache units to τ_4 (values underlined in the table), it is possible to schedule tasks τ_2, τ_3 and τ_4 together on core 1, with a CPU utilization of 0.98 and using 11 memory cache units. Then, τ_1 is scheduled to core 2. Although the algorithm still requires two cores to schedule the task set, it leaves more CPU utilization to be used on core 2 by an additional 5th task. Consequently, we can say that our second approach is able to improve the system resource usage and the schedulability. It is noteworthy to mention that for the first two sub-harmonic task sets generated, the algorithm notices that τ_1 is not schedulable along with τ_3 (using the condition of Alg. 2, line 23). Such unschedulability is shown in the table with the strikethrough text. Then, the algorithm proceeds to try to schedule the next task in the list, i.e. τ_2.

Table 3.5: Solution to Example 3.1 using HBCA2

CHOOSE SUB TASK SET FOR CORE 1												
1	τ_i	T_i	C_i	m_i	U_i	T'_i	U'_i	ΔT_i	ΔU_i	$Cum.U'_i$	$Cum.U_i$	
	τ_1	10	5	1	0.50	10	0.50	**0**	0	0.5	0.50	
	τ_3	13	6	3	0.46	10	0.60	**3**	0.14	~~1.1~~	~~0.96~~	
	τ_2	25	6	4	0.24	20	0.30	**5**	0.06	0.8	0.74	
	τ_4	25	7	4	0.28	20	0.35	**5**	0.07	1.15	1.02	
2	τ_i	T_i	C_i	m_i	U_i	T'_i	U'_i	ΔT_i	ΔU_i	$Cum.U'_i$	$Cum.U_i$	
	τ_3	13	6	3	0.46	13	0.46	**0**	0	0.47	0.46	
	τ_1	10	5	1	0.50	6.5	0.77	**3.5**	0.27	~~1.23~~	~~0.96~~	
	τ_2	25	6	4	0.24	13	0.46	**12**	0.22	0.92	0.70	
	τ_4	25	7	4	0.28	13	0.54	**12**	0.26	1.46	0.98	
3	τ_i	T_i	C_i	m_i	U_i	T'_i	U'_i	ΔT_i	ΔU_i	$Cum.U'_i$	$Cum.U_i$	
	τ_2	25	6	4	0.24	25	0.24	**0**	0	0.24	0.24	✓
	τ_4	25	7	4	0.28	25	0.28	**0**	0	0.52	0.52	✓
	τ_3	13	6	3	0.46	12.5	0.48	**0.50**	0.02	1	0.98	✓
	τ_1	10	5	1	0.50	6.26	0.80	**3.75**	0.30	1.32	1.02	
4	τ_i	T_i	C_i	m_i	U_i	T'_i	U'_i	ΔT_i	ΔU_i	$Cum.U'_i$	$Cum.U_i$	
	τ_2	25	6	4	0.24	25	0.24	**0**	0	0.24	0.24	
	τ_4	25	7	4	0.28	25	0.28	**0**	0	0.52	0.52	
	τ_3	13	6	3	0.46	12.5	0.48	**0.50**	0.02	1	0.98	
	τ_1	10	5	1	0.50	6.25	0.80	**3.75**	0.30	1.32	1.02	

3.5 Experiments, Analysis and Results

In sections 3.3 and 3.4, two approaches are proposed. It is hard to prove if one dominates the other analytically. Therefore, we use simulation results to study their performance and compare them with related work.

3.5.1 SPEC CPU2000 Benchmarks Cache Simulation

In order to test our scheduling approach, we use the data presented in [173], corresponding to the simulation results of the SPEC CPU2000 benchmarks [174] using the Simplescalar toolset [175]. The SPEC CPU2000 benchmark suite is a collection of 26 compute-intensive, non-trivial programs used to evaluate the performance of a computer's CPU,

memory system, and compilers. The benchmarks in this suite were chosen to represent real-world applications, and thus exhibit a wide range of runtime behaviors.

In order to test our algorithm, we generated a group of synthetic task sets. Each of the 26 SPEC CPU2000 benchmarks forms a curve with different points [memory size, execution time]. An exponential-fit model (with the form of $a = \exp(b)$) can thus be obtained with the 95% confidence interval values for a and b for each benchmark.

In our simulations, synthetic task sets were generated by randomly choosing a specific number of tasks n, where each task corresponds to a curve generated from the exponential-fit model of one of the 26 SPEC CPU2000 benchmarks. A thousand task sets are generated for each n. Besides, each time a new curve for a task set was generated, we used random values for a and b that fall into the 95% confidence interval of each of the two parameters.

3.5.2 Target Architecture

For the architecture in our experiments, we assume it contains a total of four cores and one cache memory, which is accessible to all cores. Similar architectures can be found commercially [176, 177]. Our cache allocation scheme may be implemented with any cache management scheme that can provide a fixed size of cache unit, and enforce strict isolation guarantees. The implementation is independent of the associativity or the replacement policy, as long as the relationship between execution times and number of cache units are given.

3.5.3 Simulation results of testing HBCA1 and HBCA2 approaches

We compare two approaches, i.e. HBCA1 and HBCA2, with three different representative scheduling schemes. The first one is the Partitioned Rate Monotonic Scheduling (P-RMS)

algorithm. This is one of the most commonly used approaches for partitioned scheduling on multi-core. A drawback for P-RMS is that it does not take the task period and execution time relationship into consideration for cache allocation and task partitioning. We use this approach as our base line approach. The second approach we investigate is the Harmonic-Fit Fixed-Priority Scheduling (HFPS) algorithm, proposed by Fan et al. [1]. This scheme takes period relationship among multiple tasks into consideration when scheduling fixed-priority tasks on multi-core platforms. Both P-RMS and HFPS do not take the variable execution times with cache allocations into consideration. Therefore, we have to use the WCET values corresponding to the worst-case scenario when $m_i = 1$. The third approach is IBRT-MCI-RMS [2] as mentioned before, which determines cache allocation based on the metric that optimizes the resource usage for a single task. These three scheduling algorithms with both HBCA1 and HBCA2 were employed to schedule the task sets on the architecture discussed above.

We define Schedulability Success Ratio (SSR) as the ratio between the number of successfully scheduled task sets divided by the total task sets tested. Figures 3.1 and 3.2 report the SSR for the tested task sets with different number of real-time tasks. Figure 3.1 shows results using a cache unit size of 1 KB. Figure 3.2 shows results using a cache unit size of 4 KB.

In Figure 3.1(a), when task number is around 14 for the case of P-RMS and 20 for the case of HFPS, we can see that the SSR starts decreasing. Also, as the number of cache units increases, as shown in Figure 3.1(b) and 3.1(c), we can see that the SSRs of P-RMS and HFPS remain almost constant. This is because they are not memory aware and therefore cannot take advantage of the increase of the number of memory cache units. On the other hand, the methods IBRT_MCI_RMS, HBCA1 and HBCA2 take advantage of the increase of the number of cache units. For instance, the schedulability success ratio

of HBCA2 starts decreasing when task number is around 45 in Figure 3.1(b) (with 256 cache units) and around 60 in Figure 3.1(c) (with 512 cache units).

In Figure 3.2(a), when the task number is around 25 for cases P-RMS, IBRT_MCI_RMS and HFPS, we can see that the SSRs start decreasing. When the task number is around 30, the SSRs start decreasing for HBCA1 and HBCA2. As the number of cache units increases (Figures 3.2(b) and 3.2(c)), IBR_MCI_RMS, HBCA1 and HBCA2 starts decreasing their SSR, for example, with task number values around 37, 49 and 58, respectively (see Figure 3.2(c)).

From the above-mentioned observations, it can be inferred that with a larger cache memory size, the memory-aware mechanisms, and especially our two approaches, are able to schedule a larger number of tasks in the system. One exception to the pattern is Figure 3.2(a). Note that this is because the number of cache units in this configuration is not large enough for the memory-aware methods to reduce the WCET values in order to increase the number of tasks schedulable in the system.

Figure 3.3 shows the schedulability of each tested mechanism with cache unit size of 1KB. Each mechanism displays the value S (maximum number of tasks such that the SSR of the evaluated method is greater than or equal to 90%) normalized against the S value obtained with P-RMS. For instance, in Figure 3.1(c), the S values for HBCA1 and HBCA2 are 47 and 62, respectively. It can be seen that HFPS always shows the same improvement, because it is a non-memory-aware mechanism. The remaining mechanisms that are memory-aware show an increasing improvement with the increment of memory cache units available per cache memory. The HBCA2 approach is able to schedule up to 4.1 times more tasks when compared to P-RMS.

Figure 3.4 shows the average values of S for data using both cache unit sizes (i.e. 1KB and 4KB) and the four cache memory sizes. From the figure, HBCA2 is able to schedule

Table 3.6: 2-Level Factorial Experiment - Factors and Levels

Factor	Name	Low	High
A	C_i and m_i init. state	$C_i = 1, m_i = 1$	λ_i (Def. 3.1)
B	Find sub task set Γ_{TS}	Simple (Alg. 1)	Enhanced (Alg. 2)
C	Cache Memory Size	256 KB	1024 KB
D	Cache Unit Size	1 KB	4 KB

up to 267% more real-time tasks than the P-RMS, and 101%, 64% and 26% more tasks when compared to HFPS, IBRT-MCI-RMS and HBCA1, respectively.

3.5.4 Full Factorial Experiment

To further study the effectiveness of the proposed algorithms, we design a 2-Level Full Factorial Design [178] in order to identify the important factors that are affecting the schedulability of the system. The four identified factors, their names and corresponding levels are shown in Table 3.6.

Fig. 3.5 shows the Pareto Chart [179] for the standardized effects, including the terms in the model up through second order. The response of the experiment is the value S, with a criterion for statistical significance, i.e. α, equal to 0.05 [178]. In general terms, the Pareto Chart shows each factor and their interaction for up to two factors. If the standardized effect for a single factor or interaction is larger than the reference line, it means that such a factor or interaction has a significant effect on the result. On the other hand, if the standardized effect of a single factor or interaction is lower than the reference line, its effect is not significant.

From Figure 3.5, we can see the significant single factors that affect the schedulability of the system. On top of the list is the total cache memory size of the architecture (C), which concludes the obvious assumption that the more memory, the higher the number of tasks able to be scheduled. The second most significant factor is the method chosen

to find the sub task set Γ_{TS} (B), which concludes that our second approach (Alg. 2) can significantly increase the schedulability of the system.

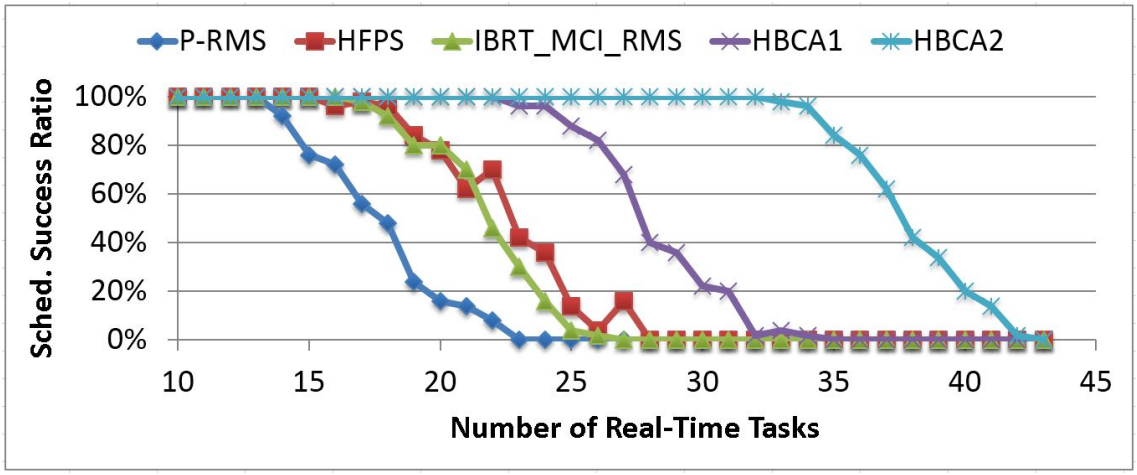
It is noteworthy of mentioning that we saw that the schedulability of the system remains constant if the high level of the factor B is used, no matter the value of the factor A. For example, this observation can be seen also in the chart realizing that the single factor (A) is also significant, but has the same effect of its interaction with the factor of finding the sub task set in the general algorithm (B). This indicates that the significance of the effect of assigning the memory cache units at the beginning of the algorithm is important, but totally conditioned to the significance of the effect of finding the sub task set Γ_{TS} using our second approach. It can be said that our second approach algorithm (high level of factor B) completely voids the effect of the utilization of the λ_i mechanism.

From Figure 3.5 we can also see that the cache unit size (D), is not significant for the schedulability of the system. This may be explained by the fact that our second approach uses the metric *CRRI* (Def. 3.5) to assign fairly the memory cache units in the system to the task which is able to decrease its CPU utilization most with the less amount of memory used. We observe that in most cases, the task that receives a small number of cache units in the first iteration of the algorithm has a very high chance to receive more cache units in the subsequent iterations. Thus, giving no effect to the granularity of memory cache units assignment.

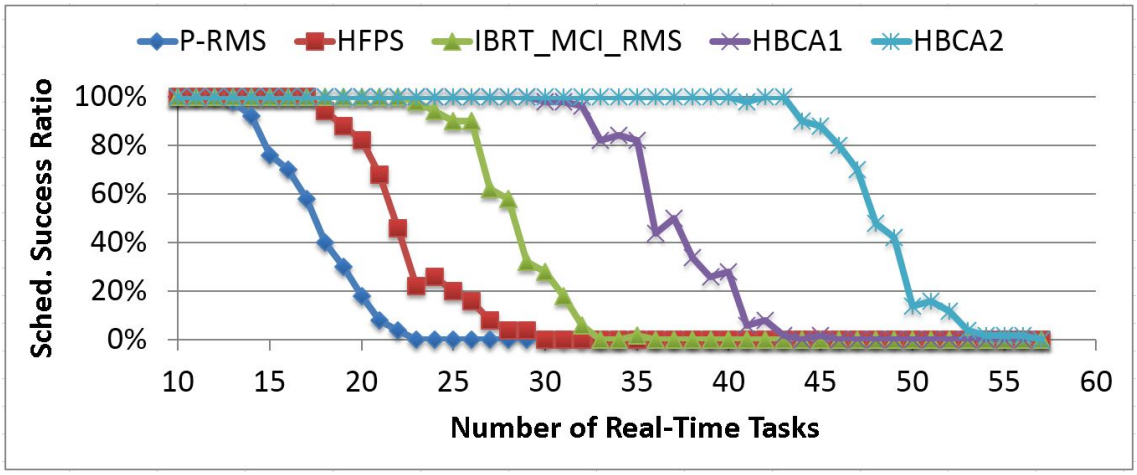
3.6 Summary

We study the cache allocation and task partitioning problem when running a set of fixed-priority real-time tasks on a multi-core platform sharing a common cache memory. We have developed two static schemes for cache allocation and task partitioning. The first one (HBCA1) combines two previous research studies that take task variable WCET times

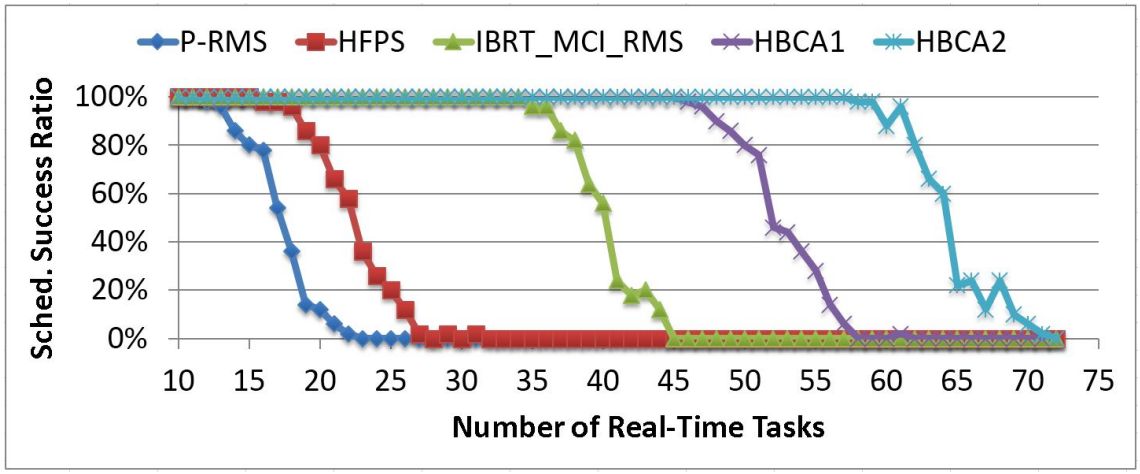
and period relationship into consideration. The second one (HBCA2) is a more elaborate approach that can judiciously choose the cache size for each task, while exploiting the task harmonic relationships within the task set. Both of them are able to successfully and significantly improve the system resource usage and the schedulability of real-time tasks, when compared with other scheduling mechanisms. Our simulation results show that our second approach increases the schedulability of real-time tasks up to four times, when compared to a conventional Partitioned Rate Monotonic Scheduling (P-RMS).



(a) Number of cache units $B = 128$

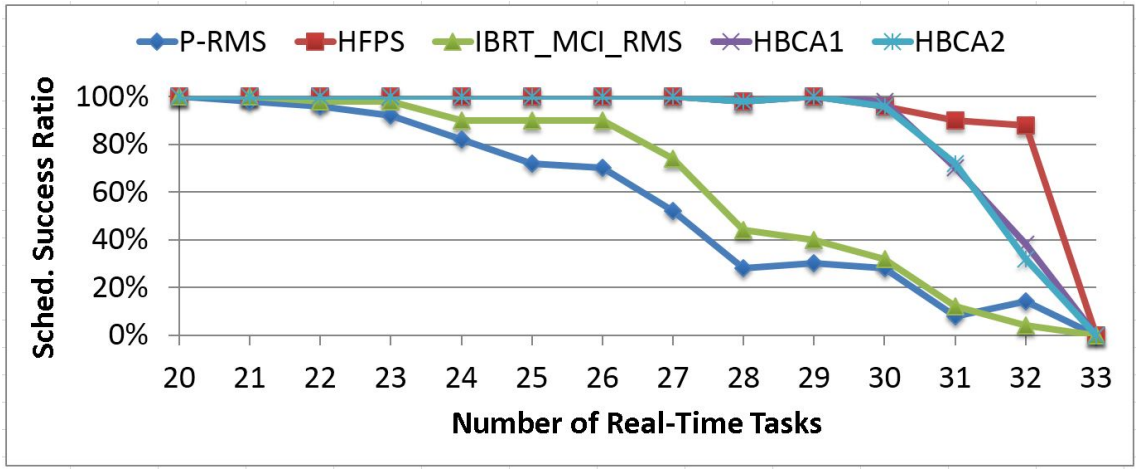


(b) Number of cache units $B = 256$

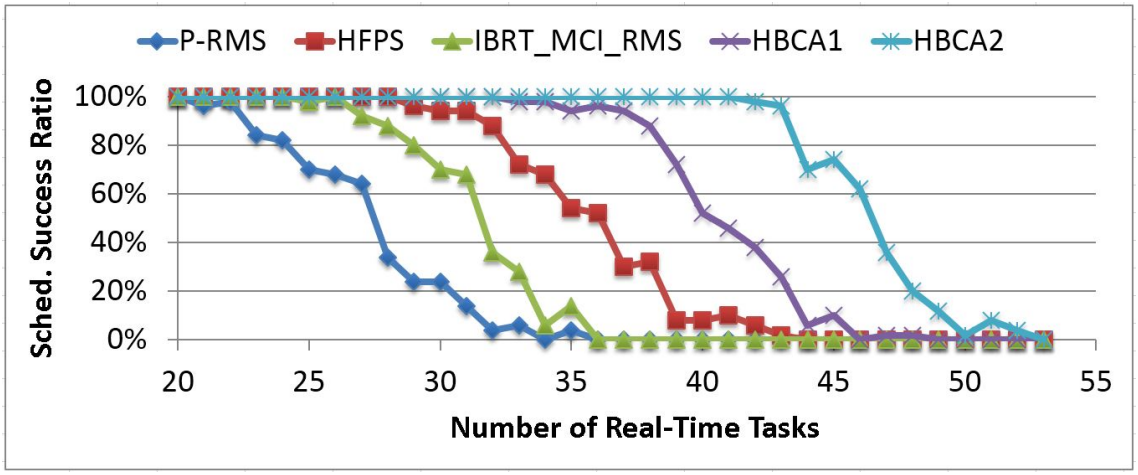


(c) Number of cache units $B = 512$

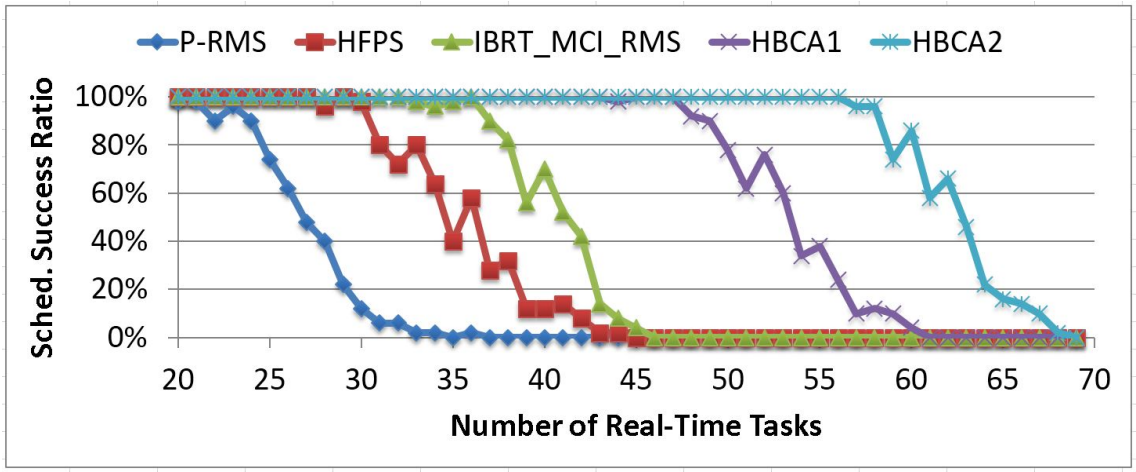
Figure 3.1: Number of Tasks VS. Scheduling Success Ratio. Cache Unit Size = 1 KB



(a) Number of cache units $B = 32$



(b) Number of cache units $B = 64$



(c) Number of cache units $B = 128$

Figure 3.2: Number of Tasks VS. Scheduling Success Ratio. Cache Unit Size = 4 KB

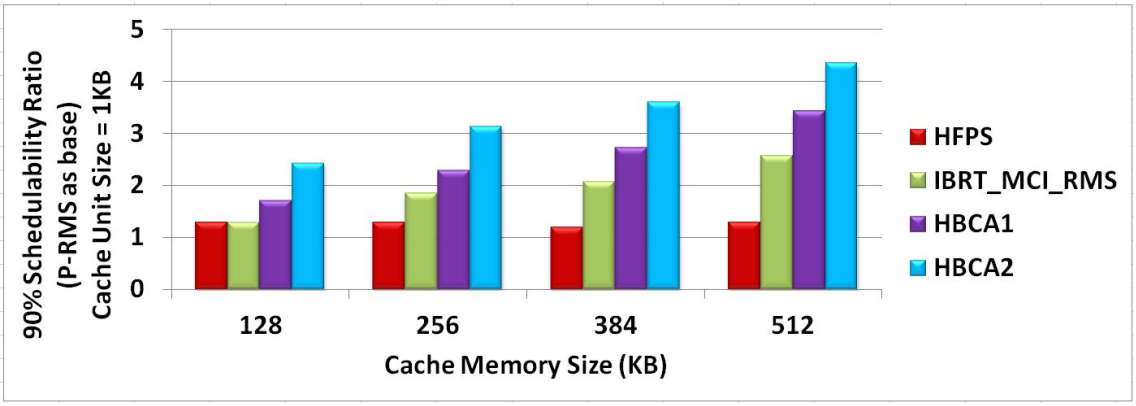


Figure 3.3: 90% Schedulability Ratio. Cache Unit Size = 1 KB

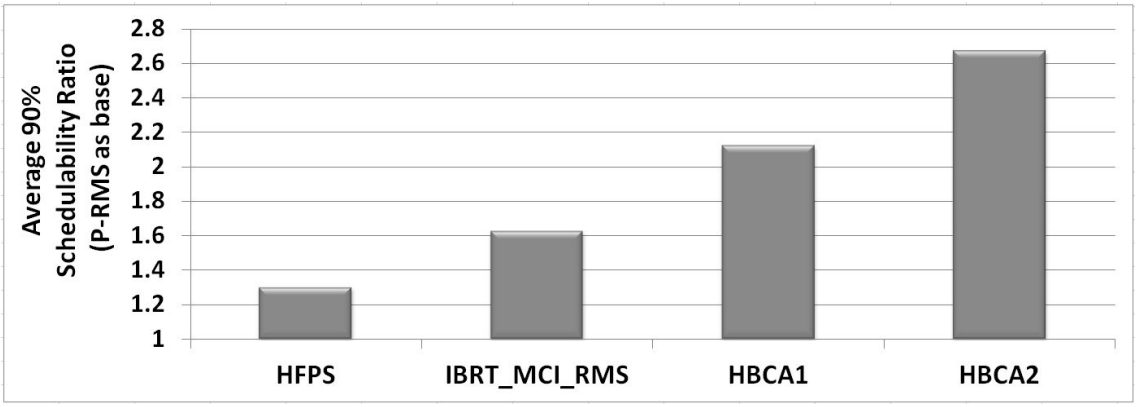


Figure 3.4: Average 90% Schedulability Ratio

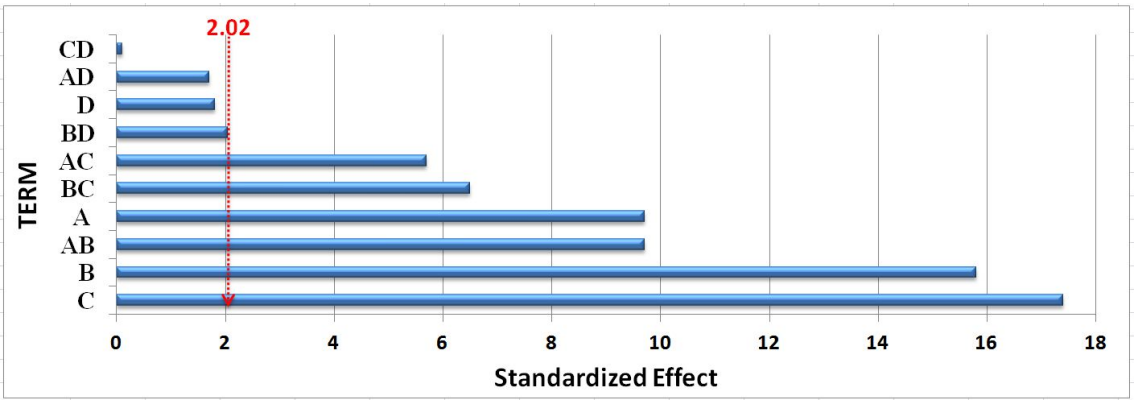


Figure 3.5: Pareto Chart of Standardized Effects (response is S with $\alpha = 0.05$)

CHAPTER 4

PROCESSOR/MEMORY CO-SCHEDULING USING PERIODIC RESOURCE SERVER FOR REAL-TIME SYSTEMS UNDER PEAK TEMPERATURE CONSTRAINTS

As stated in previous chapters, in this dissertation we study the problem of how to schedule fixed-priority real-time tasks such that they can meet their deadlines with temperatures for both the CPU and memory systems under their potentially different peak temperature limits. Thus, in this chapter, we present our research on processor and memory co-scheduling using periodic resource server considering thermal and power effects on the scheduling of real-time tasks.

The rest of the chapter is organized as follows. Section 4.1 describes the most related research projects. In section 4.2 we present our architecture and system model, the CPU and Memory power and thermal models along with our problem formulation. Section 4.3 discuss our approach in detail. We present in section 4.4 the experimental evaluation and we summarize in section 4.5.

4.1 Related Work

As previously pointed out, transistor density of processors and memory devices, as well as the memory capacity and bandwidth continue to grow, the power consumption of computing systems has also been increasing exponentially, resulting in tremendous heat generation, even to the point that threatens to disrupt the operation of the system under normal conditions [42]. An increasing chip temperature due to an excessive power dissipation has a significant impact on other design metrics, such as reliability, cost and especially on performance. Cooling down the chip temperature using mechanical methods such as cooling fans, heat spreaders, and heat sinks becomes inadequate and too expensive.

There are some techniques proposed for dynamically managing the heat generated by the memory system. For instance, one common approach is to migrate data between hot and cold devices to avoid thermal emergencies on a memory system [60]. Another approach dynamically adjusts the memory throughput to ensure that each module has a temperature below the emergency level [42]. These approaches do not take the heat generated by the CPU into account. Also, these mechanisms react to critical temperature levels and reduce or even stall the number of memory requests [18]. Consequently, such approaches can significantly affect the response time of the system and introduce an additional source of uncertainty. Thus, the excessive power consumption and heat dissipation of the memory system must be dealt with carefully. Otherwise, they can significantly affect the schedulability and predictability of real-time systems because of uncertainties introduced by different memory power/thermal management techniques. There are also research studies that consider the heat dissipation from both the CPU and memory systems, but these approaches are best-effort approaches and cannot guarantee real-time system deadlines at all [61].

In this chapter, we study the problem of how to schedule fixed-priority real-time tasks such that they can meet their deadlines with temperatures for both the CPU and memory systems under their potentially different peak temperature limits. In other words, we explore the problem of how to guarantee the feasibility of a real-time task sets under CPU and DRAM thermal constrains executed on a single-core platform. Thus, we propose an off-line mechanism to offer thermal guarantees on CPU and DRAM memory systems, for hard real-time applications, while guaranteeing also their timing constraints. We focus on fixed-priority assignment since this is the most commonly used scheme for real-time systems design in industry [23], and our proposed method can be easily extended for other scheduling methods. In our approach, we adopt the periodic resource model to manage both the processor and memory concurrently. We take advantage of the feasibility condi-

tions for a periodic resource server, established in the existing work [9], to guarantee the timing constraints for real-time tasks, and judiciously choose the periodic server settings in such a way that the peak temperature constraints for both the CPU and memory can be satisfied. To the best of our knowledge, this is the first work for thermal-aware hard real-time systems design that take the heat generations and their interactions from both the CPU and memory devices. Our experimental results, with system parameters drawn from manufacture data sheets, clearly demonstrate the effectiveness of our proposed approach in reducing the peak temperature as well as the need to take both the CPU and memory systems into considerations simultaneously for system-level thermal management.

4.2 Preliminary

In this section, we introduce the architecture and the real-time system model used in this chapter, along with the power and thermal models.

4.2.1 Architecture and System Model

In this chapter, we consider an architecture with an in-order execution single-core processor communicating to a DRAM memory of M ranks, through a single bus arbitrated by a single Memory Controller (MC), shown in Fig. 4.1. For simplicity, we assume that the DRAM MC operates the DRAM memory using a "close-row" policy and that all write memory requests are not buffered and thus consume memory bandwidth and power.

The task set consists of N independent implicit-deadline periodic tasks, denoted as $\Gamma = \{\tau_1, \tau_2, ..., \tau_N\}$, scheduled according to Rate Monotonic Scheduling (RMS). Each task τ_i, where $1 \leq i \leq N$, is characterized by its minimum inter-arrival time T_i and a WCET C_i. The maximum number of DRAM requests from any job of a task τ_i is defined as H_i.

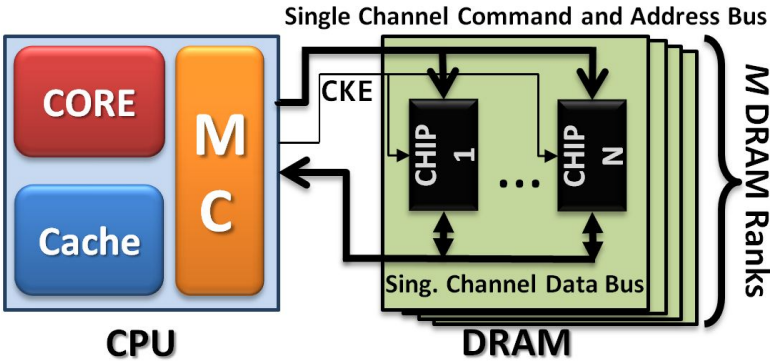


Figure 4.1: Architecture block diagram

The scheduler implements a periodic resource control mechanism (periodic server) to throttle the access of each $\tau_i \in \Gamma$ to the CPU and DRAM resources, based on the periodic resource model proposed by Shin and Lee [9] for compositional real-time systems. The reader may recall from chapter 2 that the periodic server consists of two parameters: first, the period of the server (Π), which is the recurrent time of repetition; and second, the allocation time (Θ) when the resources are available to the current scheduled task τ_i. An example schematic for a periodic server is shown in Fig. 4.2. Thus, task τ_i can not be executed or issue memory requests during the time $\Pi - \Theta$. We define OS_q as the operating system time slice, thus we assume that $OS_q \leq \Theta \leq \Pi$, with $\Pi, \Theta \in \mathbb{R}^+$ $s.t.$ $\Pi = j \cdot OS_q$ and $\Theta = m \cdot OS_q$ with $j, m \in \mathbb{N}^+$.

To ensure deadlines of hard real-time tasks in our system, we employ the schedulability condition proposed by Shin and Lee for periodic resource servers when scheduling tasks using RMS, stated in the Thm. 2.1 of chapter 2.

We assume that the CPU and DRAM systems will enter into low-power modes (see chapter 2), during the time $\Pi - \Theta$. Therefore, every time the system must exit from a low-power mode, it requires an extra overhead time Dl that must be accounted into the server allocation time Θ. Such Dl time is architecture dependent and considers delays, such as CPU and DRAM power mode changes and caches cold starts.

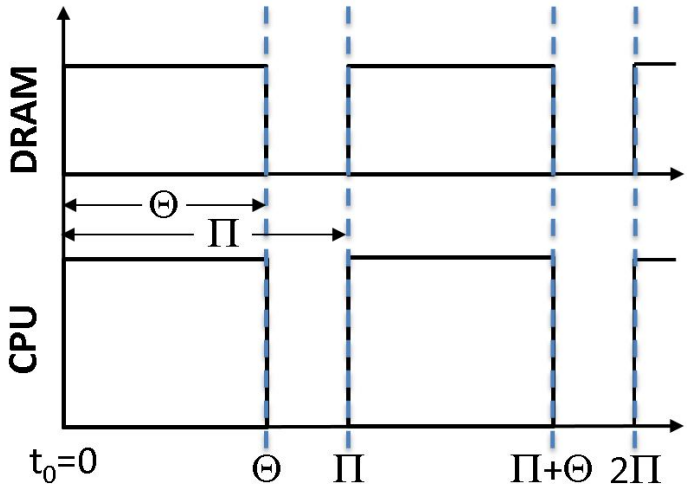


Figure 4.2: Periodic server time schedule

4.2.2 CPU and DRAM Thermal Model

For modeling the CPU system thermal behavior, we use the lumped RC model similar to Quan et al. [90], as shown in Fig. 4.3. It can be seen that the heat in the junction is dissipated through the case, a heat-spreader and then to ambient.

For modeling the DRAM system thermal behavior we use the lumped RC model similar to Ayoub et al. [180], as shown in Fig. 4.4. In this model, we see that each DRAM chip of the rank is modeled separatively with its own power source P_i^{Chip}, and a junction-to-case resistance R_{jc}, along with its thermal capacitance C_{jc}. Since the value of power for each chip is the same at any given time instant, there are no resistors in between DRAM chips. All DRAM chips will be dissipating the heat through a single heat-spreader layer, and the heat-spreader to ambient.

We consider the dependency of temperature change between both the CPU and DRAM memory systems by including a thermal resistor R_{cp} connecting both heat spreaders. A similar model is proposed, for instance in [181], for modeling the thermal interaction between layers of 3D chips, where the authors state that the large interface area between layers results in a low thermal resistance. Also, ICs connected using a silicon interposer

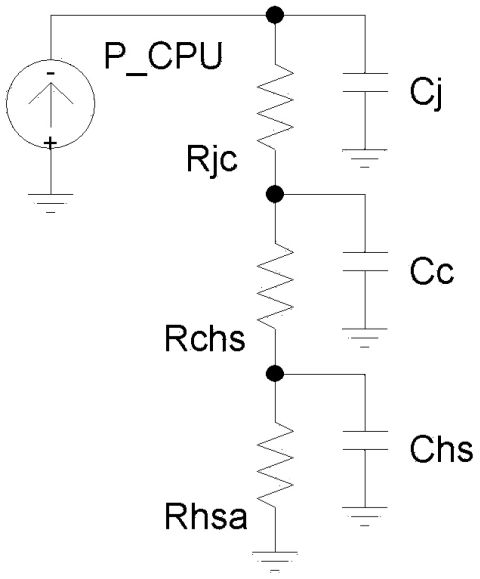


Figure 4.3: CPU Thermal Model.

(2.5D fashion) may present a considerable thermal conductance among them due to the heat dissipation of each chip through the interposer [57]. In addition, the model of Fig. 4.4 can be converted into a model similar to the one in Fig. 4.3 using superposition [180] and by making $R_{jc}^{DRAM} = R_{jc}^{Chip} \div N_{chip}$ and $C_{jc}^{DRAM} = C_{jc}^{Chip} \cdot N_{chip}$. The joint thermal model for both systems is shown in Fig. 4.5.

By applying Kirkoff's current law method to solve electrical networks, we can obtain six equations (one for each of the six nodes and not shown for space constrains).

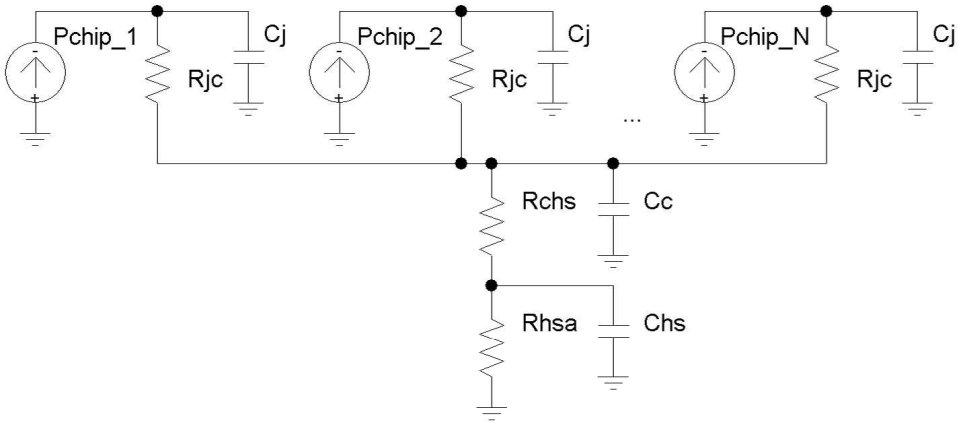


Figure 4.4: DRAM Thermal Model.

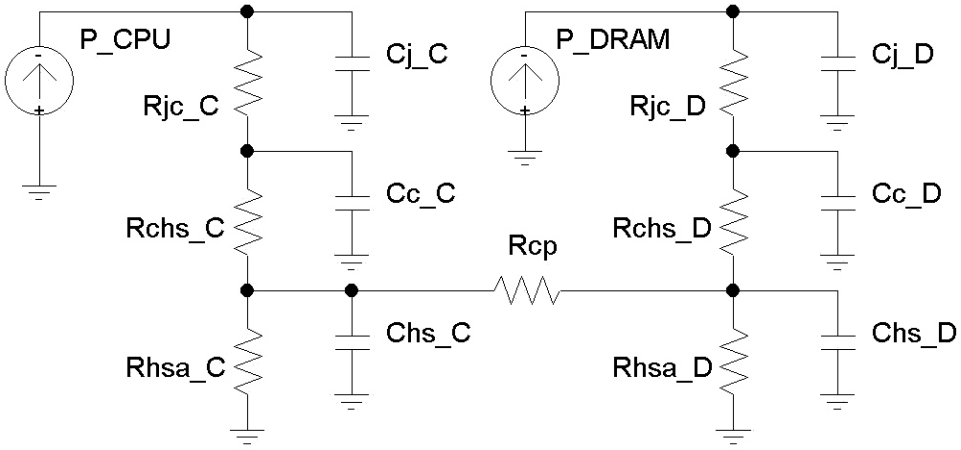


Figure 4.5: Joint CPU and DRAM Thermal Model

Considering the ambient temperature T_{amb}, the resulting equations system is:

$$\mathbf{P} = \mathbf{C} \cdot \frac{d}{dt}\mathbf{T} + \mathbf{G} \cdot \mathbf{T}(t) + \delta \cdot T_{amb}, \tag{4.1}$$

where $\mathbf{G}$ and δ are matrices dependent on the resistor values of the circuit, and $\mathbf{P} = [P_i^{CPU}, 0, 0, P_i^{DRAM}, 0, 0]$. The matrix $\mathbf{C}$ is a diagonal matrix with the values of all the six capacitances, thus it is invertible. Also, $\mathbf{P}$ can be expressed as $\mathbf{P} = \Psi + \Phi \cdot T(t)$, where Ψ and Φ are matrices containing the temperature independent and dependent portions of power, respectively. Therefore, we have the expression:

$$\frac{d}{dt}\mathbf{T}(t) = -\mathbf{C}^{-1} \cdot (\mathbf{G} - \Phi) \cdot \mathbf{T}(t) + \mathbf{C}^{-1}(\Psi - \delta \cdot T_{amb}). \tag{4.2}$$

Also, if $\mathbf{A} = -\mathbf{C}^{-1} \cdot (\mathbf{G} - \Phi)$ and $\mathbf{B} = \mathbf{C}^{-1} \cdot (\Psi - \delta \cdot T_{amb})$, the final expresion for the equations system is:

$$\frac{d}{dt}\mathbf{T}(t) = \mathbf{A} \cdot \mathbf{T}(t) + \mathbf{B}. \tag{4.3}$$

This joint system thermal model, which has a form of first-order ordinary differential equations, has a solution for $\mathbf{T}(t)$ as follows:

$$\mathbf{T}(t) = e^{t\mathbf{A}} \cdot \mathbf{T}_0 + \mathbf{A}^{-1}(e^{t\mathbf{A}} - \mathbf{I}) \cdot \mathbf{B}. \tag{4.4}$$

where T_0 is the initial temperature.

To analyze the temperature in each of the six nodes of the circuit of Fig. 4.5, we can follow the method proposed by Fan et al. [28] to analyze stable state temperature in multi-node processors. Specifically, the authors propose an analysis to calculate the temperature of any node in the circuit network, for any point in time, and for the stable state. For the sake of clarity of the following sections, we include a portion of their analysis as follows.

For any arbitrary state interval in time $[t_{q-1}, t_q]$, with κ_q the corresponding interval mode determined by the power values applied on the circuit network during such interval, once the temperatures at the starting point, i.e., $\mathbf{T}(t_{q-1})$, are given, according to Eq. 4.4, the ending temperatures of that interval, i.e., $\mathbf{T}(t_q)$, can be directly formulated as:

$$\mathbf{T}(t_q) = e^{\Delta t_q \mathbf{A}_{\kappa_q}} \cdot \mathbf{T}(t_{q-1}) + \mathbf{A}_{\kappa_q}^{-1}(e^{\Delta t_q \mathbf{A}_{\kappa_q}} - \mathbf{I}) \cdot \mathbf{B}_{\kappa_q}, \tag{4.5}$$

where $\mathbf{A}_{\kappa_q} = \mathbf{C}^{-1}(\mathbf{G}_{\kappa_q} - \Phi_{\kappa_q})$, $\mathbf{B}_{\kappa_q} = \mathbf{C}^{-1}(\Psi_{\kappa_q} - \delta \cdot T_{amb})$, and $\Delta t_q = t_q - t_{q-1}$. Note that since $\mathbf{A}_{\kappa_q}$ and $\mathbf{B}_{\kappa_q}$ are only dependent on the core running modes, i.e., κ_q, within a state interval $[t_{q-1}, t_q]$, both $\mathbf{A}_{\kappa_q}$ and $\mathbf{B}_{\kappa_q}$ are constant. Furthermore, it can be probed that the ratio between $\mathbf{B}_{\kappa_q}$ and $\mathbf{A}_{\kappa_q}$, corresponds to the temperature the system would trend to, in an infinite time, if the conditions κ_q would not change, i.e. $\mathbf{T}^{\infty}(\kappa_q) = \mathbf{B}_{\kappa_q}/\mathbf{A}_{\kappa_q}$.

4.2.3 Problem Formulation

Based on the models introduced above, our problem can be formulated as follows:

Problem 4.1. *Given a real-time task set* $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$*, where* $\forall \tau_i \in \Gamma | \tau_i = [C_i, T_i, H_i, \mu_i]$*, determine the optimal settings for a periodic server, i.e.* $[\Pi, \Theta]$*, that can ensure the timing constraints for tasks in* Γ *while keeping peak temperatures of the CPU and memory under their peak temperature constraints, i.e.* Thr^{CPU} *and* Thr^{DRAM}*, all the time.*

4.3 Our Approach

In this section, we discuss our approach in detail and present our algorithm to identify the parameters for the periodic server.

Our goal is to design a periodic server that can guarantee the deadlines for a given real-time task set, and also the peak temperature constraints for both the CPU and memory. Note that, Theorem 2.1 helps to identify the Π and corresponding allocation time Θ for a periodic server such that the timing constraints can be satisfied. The problem, however, is how to ensure the temperature constraints of the CPU and memory can be guaranteed using the periodic server. We solve this problem by attacking the following two sub-problems: (1) for a periodic server, how to bound its peak temperature; (2) how to optimize a periodic server to satisfy the peak temperature constraints for both the CPU and memory? We discuss our approach for these two sub problems below.

4.3.1 Bound the peak temperature for a periodic server

To satisfy the peak temperature constraint, one fundamental problem is to identify the peak temperature of CPU and memory for a periodic server. This may be simple for a system with a single active thermal node since, as proved by Quan et al. [90], the peak temperature must occur at one of the scheduling points. However, when dealing with both the CPU and memory, i.e. a thermal model with more than one active thermal node, to identify the peak temperature can be challenging since the peak temperature does not necessarily occur at a scheduling point, as demonstrated by Han et al. [102] and Pagani et al. [182].

When running a periodic task set on a given periodic server, there may be infinite run-time scenarios, which lead to an infinite number of power traces for both the CPU and memory. For each given power trace, Han et al. [102] and Pagani et al. [182] introduce

different methods to check its peak temperature. It is not clear, however, if a periodic server setting can guarantee that peak temperature constraints can be satisfied for all possible power traces. To this end, we define below a special power trace, called *the peak power trace.*

Definition 4.1. *Given a task set* Γ *and a periodic server* $[\Pi, \Theta]$*, let* $P_{max}^{CPU} = \max_{\tau_i \in \Gamma} \left[P_i^{CPU}\right]$ *and* $P_{max}^{DRAM} = \max_{\tau_i \in \Gamma} \left[P_i^{DRAM}\right]$*. The* peak power trace *of the periodic server is defined as the one that CPU (memory, resp.) runs at a constant power mode of* P_{max}^{CPU} *(*P_{max}^{DRAM}*, resp.) during its designated allocation time* Θ *of the periodic server.*

For a given periodic server, we can prove that the peak temperature when running the peak power trace is higher or equal than any other temperature obtained from any other possible power trace, when running the same task set. This conclusion is formulated in the following theorem.

Theorem 4.1. *Given a task set* Γ *and a periodic server* $[\Pi, \Theta]$*, the peak temperatures of CPU and memory when executing* Γ *on the periodic server are no more than the ones when running the peak power trace on the same server.*

The theorem can be easily proved by noting that at any point the power consumption when running the peak power trace is higher than that when running any other possible power traces of the same task set.

More importantly, for a peak power trace, we can quickly determine the peak temperatures for the CPU and memory, as formulated in the following theorem.

Theorem 4.2. *Given a task set* Γ *and a periodic server* $[\Pi, \Theta]$*, the peak temperatures for the CPU and memory must occur at any of the end points of any of the active intervals within one hyperperiod of the task set, i.e. the least common multiple of all task periods.*

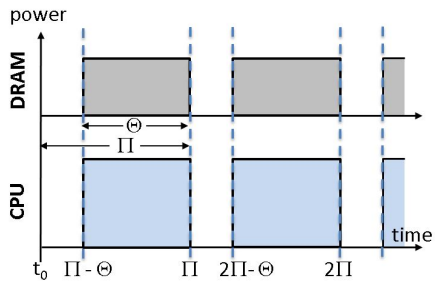


Figure 4.6: Periodic Server Time Schedule Example

Proof. The proof of this theorem is based on the analysis presented in [28, 102, 183]. Consider a periodic server schedule as seen in fig. 4.6 that has an specific values of power for CPU power and DRAM power during the time Θ, and has zero power consumption for both CPU and DRAM during the interval $[t_0, \Pi - \Theta]$. We assumed each periodic server is running with its peak power trace (see Def. 4.1). Therefore, the periodic server is running with only two possible scenarios. We have the system with only two possible values of κ_q (see Eq. 4.5), i.e. κ_{ON} for the active mode and κ_{OFF} for the power-down mode. Thus, it is possible to calculate the temperature the system would trend to, in an infinite time, if the conditions κ_{ON} and κ_{OFF} would not change. We have $\mathbf{T}^{\infty}(\kappa_{ON}) = \mathbf{B}_{\kappa_{ON}} / \mathbf{A}_{\kappa_{ON}} = \mathbf{T}_{sys}^{max}$ and $\mathbf{T}^{\infty}(\kappa_{OFF}) = \mathbf{B}_{\kappa_{OFF}} / \mathbf{A}_{\kappa_{OFF}} = T_{amb}$, with $\mathbf{T}_{sys}^{max}$ being the maximum temperature of the system if the peak power trace values were applied steadily. Then, the objective is to prove that the peak temperature must occur at any allocation time ending point, i.e. any point integer multiple of the server period Π.

The temperature $\mathbf{T}_{SS}(t)$ denotes the stable-state temperature of the nodes in the system at any instant time t. According to the following theorem proposed in the literature, it is possible to calculate such a value of $\mathbf{T}_{SS}(t)$.

Theorem 4.3. *[28] Given a periodic power trace, let $\mathbf{T}(L)$ and $\mathbf{T}(t_q)$ be the temperatures at time instant L and t_q, where $t_q \in [0, L]$, respectively. If for each eigenvalue λ_i of $\mathbf{K}$, we*

have $|\lambda_i| < 1$*, then the steady-state temperature corresponding to* t_q *can be formulated as*

$$\mathbf{T}_{SS}(t_q) = \mathbf{T}(t_q) + \mathbf{K}_q(\mathbf{I} - \mathbf{K})^{-1} \cdot (\mathbf{T}(L) - \mathbf{T}(0)). \tag{4.6}$$

where $\mathbf{K}_q = e^{\mathbf{A}\kappa_q \Delta t_q} \cdot e^{\mathbf{A}\kappa_{q-1} \Delta t_{q-1}} \cdot \ldots \cdot e^{\mathbf{A}\kappa_1 \Delta t_1}, q = 1, 2, \ldots, s.$

Using Thm. 4.3, it is possible to calculate $\mathbf{T}_{SS}(t_0)$, $\mathbf{T}_{SS}(\Pi - \Theta)$, and $\mathbf{T}_{SS}(\Pi)$. Also, note that since $\mathbf{T}_{sys}^{max}$ is the maximum temperature in the system, $\mathbf{T}_{SS}(t_0), \mathbf{T}_{SS}(\Pi - \Theta), \mathbf{T}_{SS}(\Pi) \in [T_{amb}, \mathbf{T}_{sys}^{max}]$.

According to the following theorem proposed in the literature

Theorem 4.4. *[102] Given a multi-node platform and a state interval, the temperature on each node must monotonically decrease if all the nodes' starting temperature is higher than the running mode's stable state temperature.*

and knowing that $\mathbf{T}_{SS}(t_0) \geq T_{amb}$, it is possible to conclude that the stable-state temperature of the system monotonically decreases within $[t_0, \Pi - \Theta]$.

Similarly, according to the following theorem proposed in the literature

Theorem 4.5. *[102] Given a multi-node platform and a state interval, the temperature on each node must monotonically increase if all the nodes' starting temperature is lower than the running mode's stable state temperature.*

and knowing that $\mathbf{T}_{SS}(\Pi - \Theta) \leq \mathbf{T}_{sys}^{max}$, it is possible to conclude that the stable-state temperature of the system monotonically increases within $[\Pi - \Theta, \Pi]$. In sum, the temperature monotonically decreases within $[t_0, \Pi - \Theta]$ and monotonically increases within $[\Pi - \Theta, \Pi]$, so the peak temperature must occur at Π.

□

Again, note that due to the periodicity of the peak power trace, the stable temperatures of the thermal nodes (i.e. CPU and memory) can be readily calculated using the method presented by Fan et. al in [28] (Thm. 4.3).

4.3.2 Periodic server optimization

The problem now becomes how to optimize the periodic server to meet the peak temperature constraints for both the CPU and memory. Note that, given a task set Γ, Theorem 2.1 helps to identify the upper bound for Π and corresponding allocation time Θ for a periodic server such that the timing constraints of Γ can be satisfied. In addition, with the method introduced above, we can also quickly bound the peak temperature of a given server. The problem, however, is what if the peak temperature constraints cannot be satisfied? To address this problem, we made an interesting finding, which is formulated in the following theorem.

Theorem 4.6. *Given a task set Γ and two periodic servers $[\Pi_i, \Theta_i]$ and $[\Pi_j, \Theta_j]$, let $\Theta_i/\Pi_i = \Theta_j/\Pi_j$ and $\Pi_i \leq \Pi_j$. Let $\mathbf{T}_{max}(i)$ and $\mathbf{T}_{max}(j)$ be the peak temperatures when running with the corresponding peak power traces for the two periodic servers with no power mode transition overhead, then $\mathbf{T}_{max}(i) \leq \mathbf{T}_{max}(j)$.*

Proof. The proof of this theorem is based on the analysis presented in [183]. Authors in [183] introduce the concept of m-Oscillating schedule for a periodic power trace in their following definition

Definition 4.2. *[183] Let $\mathbb{S}(t)$ be a periodic power trace schedule on a multi-node platform. The corresponding m-Oscillating schedule, denoted as $\mathbb{S}(m,t)$, is the one that scales down the length of each state interval by m times without changing its running modes κ_q.*

Note that the peak power trace periodic server of our analysis match with Def. 4.2 with $m = T_{min}/\Pi$, where T_{min} is the smallest period of any task allocated to the server, as defined in Thm. 2.1. Therefore, we can have $m_i \geq m_j$, with $m_i = T_{min}/\Pi_i$ and $m_j = T_{min}/\Pi_j$, because $\Pi_i \leq \Pi_j$.

According to the following theorem,

Theorem 4.7. *[183] Let $\mathbb{S}(t)$ be a periodic power trace schedule on a multi-node platform that contains z state intervals, $\mathbb{S}(m,t)$ be the corresponding m-Oscillating schedule, and $T_{max}(t)$ be the peak temperature of the system at any time t. Then, $T_{max}(\mathbb{S}(m,t)) \geq T_{max}(\mathbb{S}(m+1,t))$.*

we may conclude that since $m_i \geq m_j$, then $\mathbf{T}_{max}(i) \leq \mathbf{T}_{max}(j)$. □

According to Theorem 4.6, if a periodic server cannot meet the temperature constraints, we can always try to minimize its peak temperature by reducing the server period while keeping the capacity, i.e. Θ_j/Π_j, to ensure the timing constraints. From Theorem 4.6, the smaller the period, the lower the peak temperature. However, this conclusion is true and can be proved only when the power mode transition overhead can be ignored. In practical scenarios, when the CPU or memory transit from one power mode to another, the system incurs not only a timing penalty but also a power penalty. The more the transitions happen, the larger the total overhead becomes and compromises the potential improvement. In our approach, we resort to a simple search algorithm to find the trade-offs between the increased transition overhead and server period reduction to maximize the peak temperature reduction.

4.3.3 CPU/Memory Co-Scheduling using Periodic Server (CSPS)

We are now ready to present our CPU/memory co-scheduling algorithm using periodic server, developed based on Theorems 2.1, 4.1, 4.2 and 4.6, as shown in Alg. 3.

Alg. 3 takes inputs including task set Γ, the operating system time slice (OS_q), the temperature thresholds for the CPU and DRAM, and the server overhead time Dl. For a given value of the server period Π_x, the algorithm finds the minimum allocation time Θ_{min} that guarantees the timing constrains of Γ according to theorem 2.1 (line 6). The overhead time is further accounted into the allocation time (line 7). Then, with the server

period, allocation time, and power values of CPU and DRAM (P_{max}^{CPU} and P_{max}^{DRAM} for time Θ, P_{S0}^{CPU} and P_{SR}^{DRAM} for time $\Pi - \Theta$), the algorithm generates the peak power traces for the CPU and DRAM (lines 8 and 9, respectively). The peak temperatures for CPU and for DRAM in stable state are calculated using Theorem 4.2 (line 10), then compared against the corresponding thresholds (line 11). If the peak temperatures are both under the thresholds, then the parameters Π_x and Θ_x are stored in a matrix Λ (line 16).

Algorithm 3 iteratively tests all possible values of Π_x between a maximum and a minimum value (*for loop* line 5), seeking to obtain a lower peak temperature bound, following Theorem 4.6. According to Theorem 2.1, it is necessary that the period for the server must be lower than the minimum period of any $\tau_i \in \Gamma$, i.e. $\forall i, 1 \leq i \leq N, \Pi_i \leq T_{min}$. Also, the minimum value or the server period must be the operating system time slice. Thus, $OS_q \leq \Pi_x \leq T_{min}$ (lines 1 and 2). According to Theorem 4.1, if there exists at least one single setting $[\Pi, \Theta]$ in the set of feasible combinations Λ, then it is possible to conclude that Γ is safely feasible and schedulable under the CPU and DRAM temperature constrains, and the algorithm chooses the schedulable combination Λ_{sch} such that the overall system temperature is minimum as the final output. If Λ is empty, then we cannot conclude that Γ is feasible (lines 19 to 23).

4.4 Experiments, Analysis and Results

To study the effectiveness of our proposed approach, we compared the simulation results of our approach with other related approaches listed below:

- *No-Server* This is the most primitive approach. CPU/memory runs a task when the task queue is not empty and enters low power modes when no task is ready. No periodic server is applied in the scheduler. This is the traditional dynamic power-down approach.

Algorithm 3 Co-Scheduling using Periodic Server (CSPS)

Input: $\Gamma = [C, T, H, \mu]$, OS_q, Thr^{DRAM}, Thr^{CPU}, Dl
Output: Whether Γ is feasible or not, and Λ_{sch}

1: $\Lambda = \emptyset$; $\Pi_{min} = OS_q$; $\Pi_{TS} = \infty$; $\Theta_{TS} = \infty$;
2: Π_{max} = minimum period of any $\tau_i \in \Gamma$;
3: $P_{max}^{CPU} = \max_{\tau_i \in \Gamma} \left[P_i^{CPU}\right]$;
4: $P_{max}^{DRAM} = \max_{\tau_i \in \Gamma} \left[P_i^{DRAM}\right]$;
5: **for** $\Pi_x = \Pi_{max}$ downto Π_{min} with $step = OS_q$ **do**
6: $\Theta_{min} = MinimumAllocationTime(\Pi_x, \Gamma)$;
7: $\Theta_x = \lceil (\Theta_{min} + Dl) \div OS_q \rceil \cdot OS_q$;
8: $\mathbf{P_x^{CPU}} = PowerTraceCPU(\Theta_x, \Pi_x, P_{max}^{CPU}, P_{S0}^{CPU})$;
9: $\mathbf{P_x^{DRAM}} = PowerTraceDRAM(\Theta_x, \Pi_x, P_{max}^{DRAM}, P_{SR}^{DRAM})$;
10: $(Tp_x^{CPU}, Tp_x^{DRAM}) = SystemTemperature(\mathbf{P_x^{CPU}}, \mathbf{P_x^{DRAM}})$;
11: **if** $Tp_x^{CPU} \leq Thr^{CPU}$ **AND** $Tp_x^{DRAM} \leq Thr^{DRAM}$ **then**
12: $\Pi_{TS} = \Pi_x$;
13: $\Theta_{TS} = \Theta_x$;
14: $Tp_{TS}^{CPU} = Tp_x^{CPU}$;
15: $Tp_{TS}^{DRAM} = Tp_x^{DRAM}$;
16: $\Lambda = \Lambda \bigcup \{Tp_{TS}^{CPU}, Tp_{TS}^{DRAM}, \Pi_{TS}, \Theta_{TS}\}$;
17: **end if**
18: **end for**
19: **if** $\Lambda \neq \emptyset$ **then**
20: $\Lambda_{sch} = \{Tp^{CPU}, Tp^{DRAM}, \Pi, \Theta\} \in \Lambda$, s.t. overall system temperature is minimum;
21: **else**
22: Return: Γ is not schedulable;
23: **end if**

- *CPU-Only:* This represents many existing approaches (such as [46]) that ignore the thermal impacts of memory. CPU is managed using a periodic server, with server parameters $[\Pi, \Theta]$ determined by the heat dissipation of CPU only;
- *Mem-Only:* This approach ignores the thermal impacts of CPU. CPU is managed using a periodic server, with server parameters $[\Pi, \Theta]$ determined by the heat dissipation of memory only;
- *CPU/Mem-Co-Scheduling:* This is our approach illustrated in Algorithm 3.

In our simulation, we adopted the similar CPU parameters used in previous works, such as [28, 90, 106]. We built our memory model based on the Micron's power model. Specifically, we adopted the power model for the DDR3 DRAM chip of 2GB, using a low conductivity substrate [184], for a memory system with a total of four ranks ($M = 4$). Based on this model, we observed that the maximum DRAM power is approximately half of the maximum CPU power, which fits the power ratio of CPU and memory reported in the literature [59]. We assumed an ambient temperature value of $35°C$.

We randomly generated 3000 task sets for our simulation. Specifically, we randomly generated task periods and execution times such that the task set utilizations cover a range from 0.01 to 0.69. We randomly generated values for the parameter μ_i in Eq. 2.5 within specific ranges to generate different types of tasks: High-memory bounded ($[0.8, 1.0]$), mild-memory bounded ($[0.6, 0.8]$), and CPU-bounded ($[0.01, 0.6]$). Also, using the same ranges, we randomly generated the ratio between each task maximum number of memory requests and the maximum memory bandwidth, i.e., H_i / BW_{max}^{DRAM}. For overhead, we chose a value of 1 ms which is able to account for CPU and memory wake ups, plus cache cold starts [185]. We assume an $OS_q = 5ms$, which is a common value among operating systems.

We set the peak temperature threshold for CPU with the typical value of 90°C. Also, we set the memory peak temperature threshold as the memory datasheet nominal maximum value of 85°C. Additionally, we compare also with a memory peak temperature threshold value of 60°C. Although such a temperature is below the datasheet maximum ratings, according to recent studies [186], it is possible to further improve the retention time and achieve additional power and energy savings in a DRAM system if memory temperature is low.

Fig. 4.7 compares the feasibility of the tested task sets using each of the different approaches and setting the DRAM peak temperature threshold to 85°C. Fig. 4.7a compares high-memory bounded tasks, 4.9b and 4.7c compare mild-memory bounded and CPU-bounded tasks, respectively. Additionally, Fig. 4.8 shows the average of these three sub-figures. Each sub-figure shows feasibility results using different values for the coupling thermal resistor R_{cp} of Fig. 4.5, with values equal to 0.1 K/W, 1 K/W and 10 K/W. Also Fig. 4.9 compares the feasibility of the tested task sets like Fig. 4.7, but setting the DRAM peak temperature threshold to 60°C.

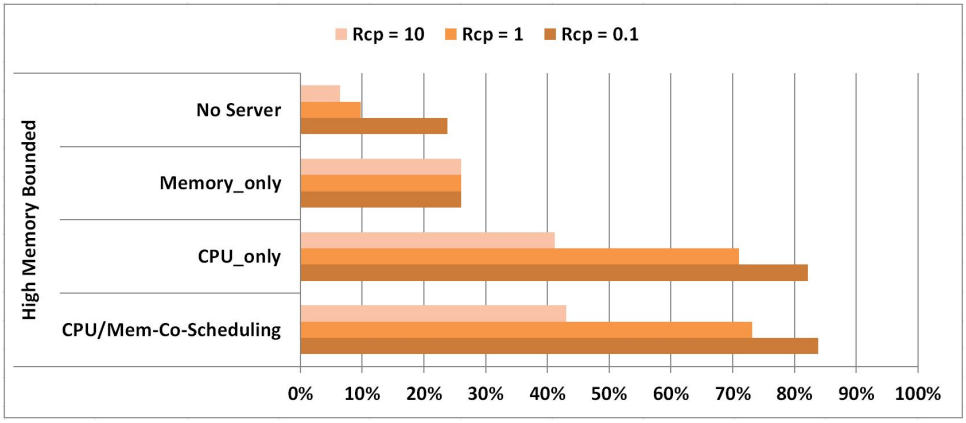
When comparing the results of the *No-Server* approach, we can immediately see the significant improvement of task set feasibility when using a periodic server. All three sub-figures of Fig. 4.7 show that the *No-Server* approach has less than 40% schedulability than the other three approaches implementing a periodic server. Also, the schedulability of the *Memory-only* approach decreases with a higher memory usage, which is a logical result since with a higher memory usage the power consumption of DRAM is increased and therefore its operating temperature.

Additionally, the schedulability of the *CPU-only* approach is the lowest for CPU-bounded tasks (Fig. 4.7c). However, the schedulability of *CPU-only* for the case of mild-memory bounded tasks is higher than for the case of high-memory bounded tasks. This can be explained in the fact that such type of workload is generating a balanced

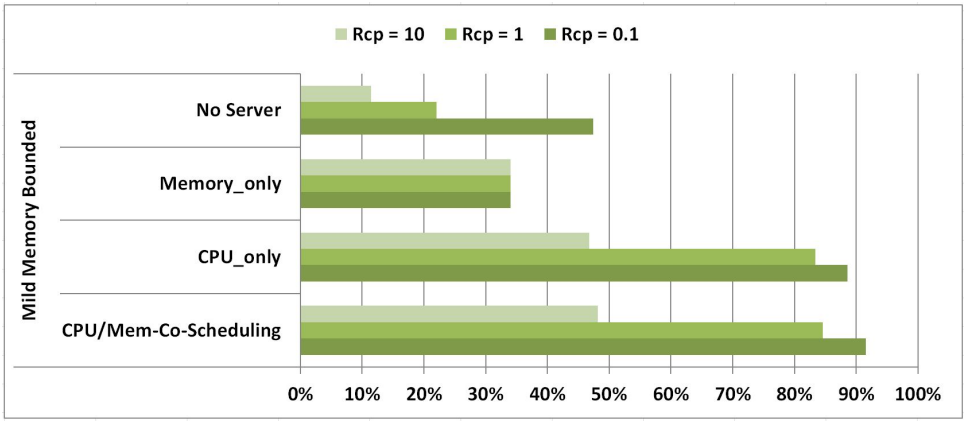
power consumption of CPU and DRAM that allows both systems to reduce their peak temperatures.

When comparing the *CPU/Mem Co-Scheduling* approach with *Memory-only* and *CPU-only*, for the cases of mild-memory bounded and high-memory bounded tasks, our approach *CPU/Mem Co-Scheduling* always shows a higher schedulability for all the three different values of coupling resistor R_{cp}. The increment in schedulability for the *CPU/Mem Co-Scheduling* approach is around 3% when compared to the *CPU-only* approach, and more than 40% in average, when compared to the *Memory-only* approach. Fig. 4.7a and 4.7b show that for some cases it is necessary to consider the parameters from both the CPU and memory sub-systems to generate an schedule that guarantee the timing and temperature constraints of the system. When comparing the *CPU/Mem Co-Scheduling* approach with the *CPU-only* approach, for the case of CPU-bounded tasks, it can be seen that our approach has almost the same schedulability values as *CPU-only* (see Fig. 4.7c). This can be explained by the fact that for these type of tasks the major power consumption is performed by the CPU, and the CPU will drive the system's peak temperatures.

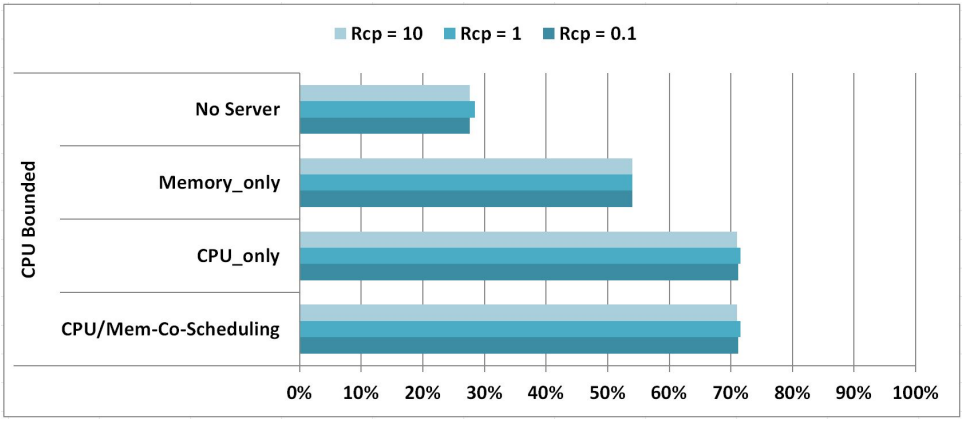
Fig. 4.8 shows the average feasibility values for the three types of task sets considered (from high-memory bounded to CPU-bounded) when the DRAM peak temperature threshold is set to $85°C$. It can be seen an increment of feasibility when using our proposed methodology, compared to the other three approaches. Also notice that when the value of R_{cp} is small, since both the CPU and memory sub-systems are highly coupled, the memory power can be dissipated more efficiently through the CPU cooling platform, lowering the overall systems peak temperatures. As can be seen, our methodology is able to identify real-time task workloads where the dependency of either the CPU or the memory makes the system unschedulable under peak temperature constraints, if the server parameters are analyzed considering only the CPU characteristics or considering only the memory characteristics.



(a)



(b)



(c)

Figure 4.7: Task set feasibility comparison using each different method, for different types of tasks with DRAM peak threshold temperature of 85°C. High memory bounded (a), mild memory bounded (b) and CPU bounded (c), using also different values for R_{cp} of Fig 4.5.

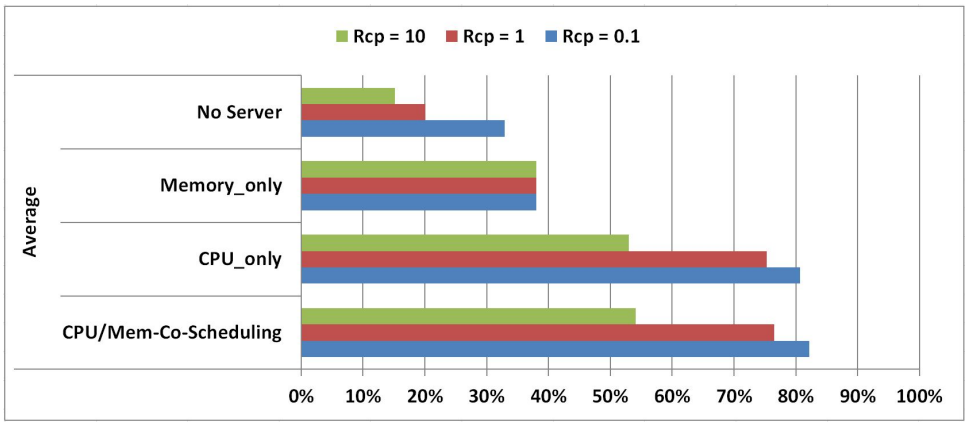
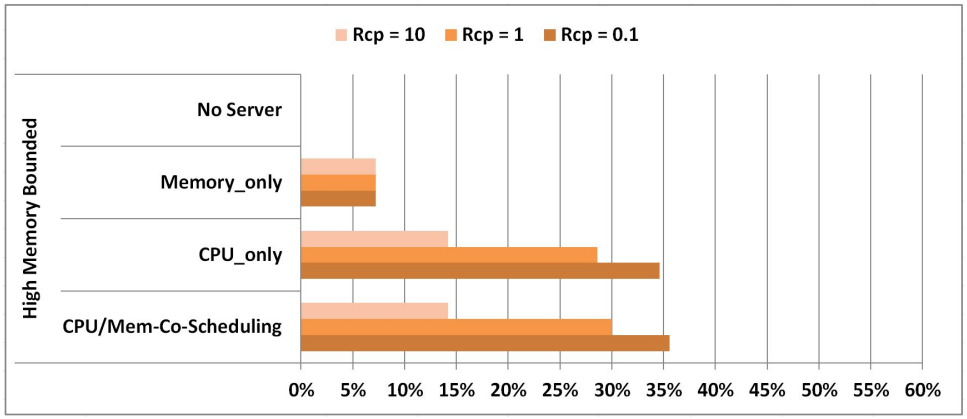


Figure 4.8: Average task set feasibility comparison with CPU peak threshold temperature of 90°C and DRAM peak threshold temperature of 85°C

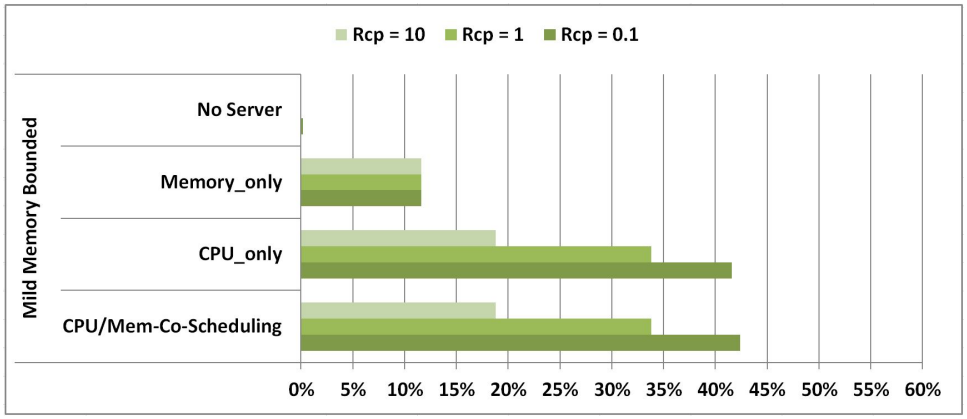
Fig. 4.9 shows similar results as Fig. 4.7, but with reduced values in the schedulability of tasks sets using any method. Since the peak temperature threshold for DRAM is set to 60°C, the majority of the resulting feasibility values are conditioned by the schedulability of the DRAM device. Notice that even for the case of CPU-bounded tasks, our proposed approach is able to increase the feasibility of the system (Fig. 4.9c).

Fig. 4.10 shows the average feasibility values for the three types of task sets considered (from high-memory bounded to CPU-bounded) when the DRAM peak temperature threshold is set to 60°C.

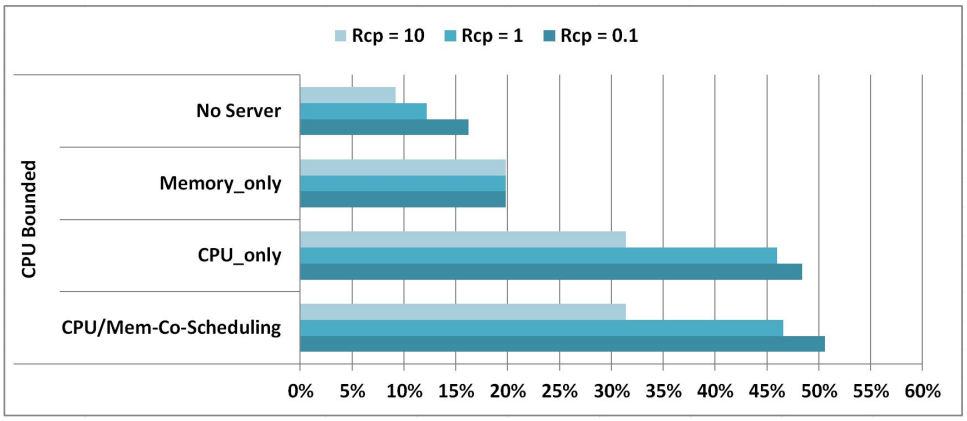
We can confirm our previous conclusion above mentioned, i.e., that when CPU and memory are tightly coupled within a small space, and tasks are mixed between memory-bounded and CPU-bounded ones, the heat dissipation generated from the memory can be more effectively dissipated using the cooling methods for CPU, at a cost of increased CPU temperature. If the peak temperature threshold of the DRAM device must be set to a low value in order to extend the retention time and minimize the number of refreshes, it is still possible to implement a periodic server in the platform that guarantee a higher task set feasibility, but the DRAM peak temperature will drive mostly the feasibility condition of a particular task set, with a task set feasibility decrement.



(a)



(b)



(c)

Figure 4.9: Task set feasibility comparison using each different method, for different types of tasks with DRAM peak threshold temperature of 60°C. High memory bounded (a), mild memory bounded (b) and CPU bounded (c), using also different values for R_{cp} of Fig 4.5.

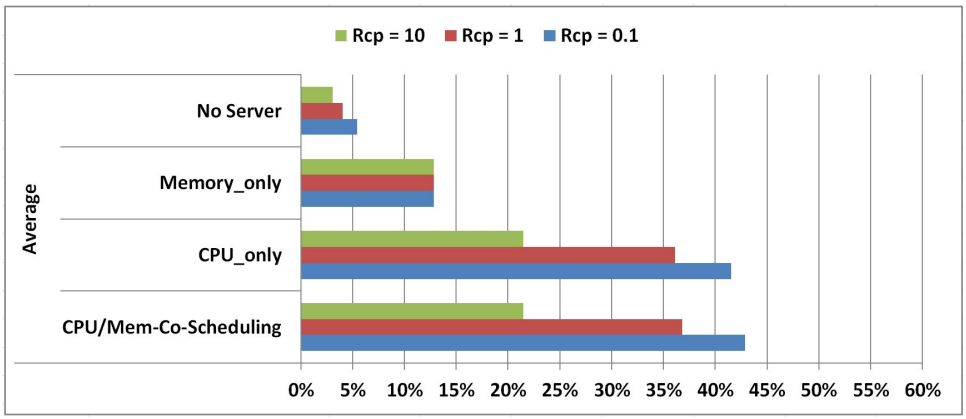


Figure 4.10: Average task set feasibility comparison with CPU peak threshold temperature of 90°*C* and DRAM peak threshold temperature of 60°*C*

Thus, as our experimental results demonstrate, as memory power consumption becomes more and more significant, an effective thermal solution needs to handle the heat dissipation not only from CPU but also from memory as well.

4.5 Summary

Thermal management is critical for many IoT applications that exhibit real-time characteristics with high power densities. We develop a novel strategy that employs periodic server to collaboratively schedule both the CPU and memory to meet the timing constraints for a real-time task set and in the meantime satisfy the temperature constraints for both the CPU and memory. To the best of our knowledge, this is the first work for thermal-aware hard real-time system design that takes into account the heat generations and their interactions from both the CPU and memory devices. Our experimental results, with system parameters drawn from manufacture data sheets, clearly demonstrate the effectiveness of our proposed approach in reducing the peak temperature as well as the need to take both the CPU and memory systems into consideration simultaneously for system-level thermal management.

CHAPTER 5

THERMAL-AWARE JOINT CPU AND MEMORY SCHEDULING FOR HARD REAL-TIME TASKS ON MULTICORE 3D PLATFORMS

As introduced in chapter 1, it is expected that in the near future multiple industries begin to develop commercially real-time systems using 3D integrated platforms, and given the dramatically increased power density not only from CPUs but also from memory systems as well, which are expected to consume as much power as the processing cores in embedded 3D platforms [154], we believe that a joint CPU and memory system resource management is highly desired for 3D platforms to effectively deal with the heat dissipation confined in a small package. In addition, different from many existing thermal management strategies, which are reactive and best-effort in nature, we are more interested in ones that can ensure the strong guarantee for real-time applications.

Thus, in this chapter we study the problem of how to schedule a set of fixed-priority hard real-time tasks on a 3D multi-core platform, while keeping temperatures for both the logic layer and memory layers under peak temperature limits. Again, we focus on a fixed-priority assignment since this is the most commonly used scheme for real-time systems design in industry [23, 77]. Additionally, our proposed method can be easily extended to other scheduling methods.

A key challenge in our problem is how to manage CPUs and memory systems in a collaborative manner to meet the task's timing requirements and peak temperature constraints. There have been many thermal aware resource management strategies proposed on 3D architectures, e.g. [155–157, 187]. Most of them are reactive and best-effort in nature. Thus, these approaches are intuitive and may be effective in dealing with thermal emergencies. However, one major problem is that they cannot guarantee that the chip's temperature will not exceed pre-determined thresholds.

To overcome this challenge, we present a novel approach that incorporates the periodic resource model [9] for both CPUs and memory systems to guarantee the timing constraints for hard real-time systems under thermal constraints. The periodic resource model can accurately capture the characteristics that guarantee resource allocations of Θ time units every Π time units [9] [80]. The beauty of such a model is that it allows us to take advantage of the feasibility conditions for a periodic resource server (see chapter 2), to guarantee the timing constraints for real-time tasks. More importantly, we can also take advantage of the periodic behaviors of CPU and memory and formulate the temperature dynamics analytically to achieve the deterministic guarantee for thermal constraints. Our approach chooses judiciously the periodic server settings for each processing core and for the memory bus arbitration in such a way that the peak temperature constraints for both the multi-core layer and memory layers are satisfied. To the best of our knowledge, this is the first work for thermal-aware hard real-time systems design implemented on a 3D platform that considers the heat generations and their interactions from both the multi-core and memory layers. Our experimental results, with system parameters drawn from manufacturer's data sheets, clearly demonstrate the effectiveness of our proposed approach in increasing the schedulability of real-time tasks, while keeping the peak temperature of the system under a threshold value.

The rest of the chapter is organized as follows. Section 5.1 describes the most related research projects. In section 5.2 we present our architecture and system model, the CPU and memory power and thermal models. Section 5.3 describes our problem formulation and discuss our approach in detail. We present in section 5.4 the experimental evaluation and we summarize in section 5.5.

5.1 Related Work

3D integration technology is a way to mitigate the "Memory Wall" challenge in future microprocessors [153]. This sharp rise in temperature prohibits the full potential of 3D stacks even though they have much higher bandwidth and thus capability to handle higher loads. In other words, 3D stacks hit the thermal wall at higher frequencies prohibiting performance scaling. It should be noted that the rise in temperature is caused by the higher activity of the cores which can occur due to various reasons, such as, higher frequency, complex cores, compute-intensive applications, power-hungry accelerators, etc. Furthermore, the increase in temperature decreases the charge retention capability of DRAM increasing refresh rate, and also has a significantly negative impact on DRAM lifetime and reliability. As a result, it has been advised to use 3D stacks only for memory-intensive applications running at a lower frequency [53], diminishing the full potential of 3D integration technology.

3D memory-processor integration has received considerable attention in the literature [151–153], and multiple research studies have been proposed to manage the thermal problems in 3D integration technology. For instance, Meng et al. [128] introduces a framework to model on-chip DRAM accesses and analyzes performance, power, and temperature trade-offs of 3D systems. Their architecture focuses on one single layer of logic and one layer of DRAM memory. Chen et al. [154] characterize the thermal and performance behavior of the target architecture when the voltage and frequency levels of cores and DRAMs are synergistically controlled, targeting an architecture with multiple layers of DRAM memory. Some studies propose to manage the power and thermal parameters in 3D ICs by performing memory mapping techniques [155, 156]. The research proposed in [188] describes a novel runtime system that scales the frequency of both processor and DRAM-based on the performance and power models. Authors model

the system power consumption at various processor and DRAM frequencies to find opportunities for scaling the voltage and frequencies of both CPU and DRAM in order to minimize the energy consumption. Other thermal management approaches for 3D architectures, such as [157], propose to reduce temperature variance and the peak temperature of a 3D multi-core processor and stacked DRAM by thermally-aware thread migration among processor cores. However, these thermal management mechanisms are online reactive in nature, and cannot ensure real-time guarantee under thermal constraints on 3D platforms.

There are research works (e.g. [80]) that offer mechanisms to guarantee the schedulability of real-time systems on multi-core platforms, employing periodic resource servers, but without considering the power and thermal properties of the applications. There are also a few research works, such as [28] that consider the power and thermal properties of the system, but they only consider processing cores neglecting the DRAM power impact on the whole system temperature. Research works, such as [45,46] propose mechanisms to minimize the peak temperature of the CPU using periodic resource models. However, these works do not consider the effect of the number of memory requests of each task, nor the power and temperature of the DRAM system. Additional works such as [189] and [190], propose thermal management mechanisms, considering only the thermal interaction of adjacent layers in a 3D architecture, neglecting the thermal relation between areas within the same layer, due to the high thermal coupling between layers. These works use the concept of super-core, where all the architecture areas vertically adjacent are treated as a single thermal unit in order to manage its temperature. However, their methodologies are aimed to be implemented on 3D logic-to-logic integration.

Other research works, such as [191], consider the power-performance trade-off between memory and CPU, but do not consider the thermal behavior of the system. A closely related work is proposed in [187]. Authors propose a thermal-aware task allo-

cation, memory mapping, and task scheduling methodology for the 3D stacked memory and processor architecture in order to reduce the system's peak temperature. However, this research work does not specifically address the timing guarantees of hard-real time task. Unlike their mechanism for reducing the power consumption of the memory layers only, our methodology proposes to use periodic resource servers to synergistically control the power of logic and memory layers.

5.2 Preliminary

In this section, we introduce the architecture and the real-time system model used in this paper, along with the power and thermal models.

5.2.1 System Architecture

In this work, we consider a 3D system architecture, similar to that in [192], consisting of a logic layer with multiple DRAM layers on top of it. We assume face-to-back bonding and inter-layer communication using through-silicon vias (TSVs) that are etched through the bulk silicon for vertically connecting the layers [150].

The logic layer consists of P in-order execution processing cores denoted as P_k, with $k = 1, 2, ..., P$, each one with its own L1 instruction and data caches. Each core also has a private L2 cache. We further assume that such a logic layer is the closest to the heat sink.

The DRAM layers consist of Y different layers. Each DRAM layer consists of B symmetrical banks. For this work, we assume that $P = B$, such that each core P has Y banks on top of it. In total, the platform has L layers (memory plus logic), where $L = Y + 1$. We also assume that each set of banks of different DRAM layers comprises a DRAM rank. Thus, the system has B different ranks, each of which has Y banks. Additionally, the number of ranks is equal to the number of processing cores, and each

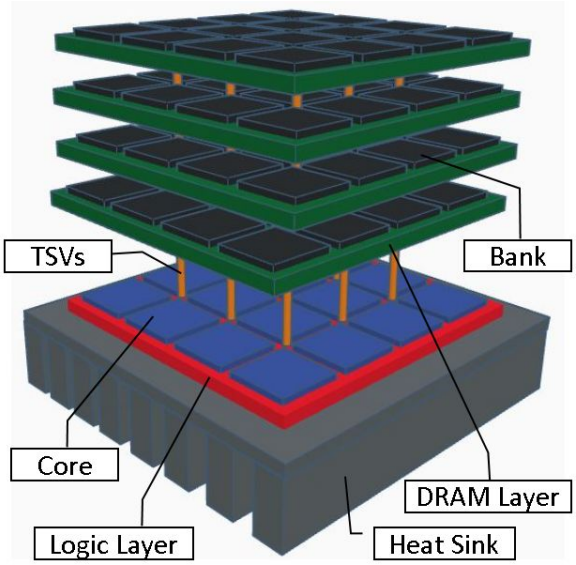


Figure 5.1: 3D platform example with 16 cores in one logic layer and 16 banks per DRAM layer in 4 memory layers

pair is vertically aligned. We consider the system to be implemented using memory-partitioning, such that each processing core P_k can issue memory requests only to the rank B_k on top of it, and such memory requests will be equally distributed among all the banks of the same rank. An example 3D architecture with one logic layer holding 16 processing cores and 4 DRAM layers with 16 DRAM banks each is shown in Fig. 5.1.

The logic layer also has Q on-chip memory controllers (MC), denoted as MC_q with $q = 1, 2, ..., Q$, to interface with the different DRAM ranks. We do not make any assumption on the number of MCs in the system. However, since the timing constraints depend on the task execution time that is affected by the memory delay, we assume the memory accesses to each rank of each processing unit are bound by the implementation of a TDMA bus arbitration [193] on each memory controller in the platform. Normally the power of each MC is a constant [128] [154], so in this work, we do not consider the effect of the power consumed by each MC in the system's peak temperature.

For simplicity, we assume that the MC operates the DRAM ranks using a "close-row" policy, and that all write memory requests are not buffered and consume memory

bandwidth and power. Furthermore, we assume that each memory transaction consumes the same amount of power in the memory.

5.2.2 System Model

Task Set Model

The task set consists of N independent implicit-deadline periodic tasks, denoted as $\Gamma = \{\tau_1, \tau_2, ..., \tau_N\}$, scheduled according to the Rate Monotonic Scheduling (RMS). Each task τ_i, where $1 \leq i \leq N$, is characterized by its minimum inter-arrival time T_i and a worst-case execution time (WCET) C_i when running in isolation in the system with all the DRAM Bandwidth available (BW^{max}). The maximum DRAM BW used by any job of a task τ_i is defined as H_BW_i, and the minimum DRAM BW from any job of a task τ_i is defined as Z_BW_i. We consider a system with static partitioning, i.e., each processing core is assigned with a fixed subset of tasks Γ_k.

Periodic Resource Model

We consider that the scheduler implements a periodic resource control mechanism (periodic server) on each processing core P_k to throttle the access of each $\tau_i \in \Gamma_k$ to the corresponding core resource, based on the periodic resource model proposed by Shin and Lee [9] for compositional real-time systems. In our implementation, the periodic server consists of two parameters: first, the period of the server (Π_k), which is the recurrent time of repetition; second, the allocation time (Θ_k) when the resources are available to the currently scheduled task $\tau_i \in \Gamma_k$. Hence, we have that $\Theta_k \leq \Pi_k$. Thus, task τ_i cannot be executed during the time $\Pi_k - \Theta_k$. A schematic of a periodic server is shown in Fig. 5.2.

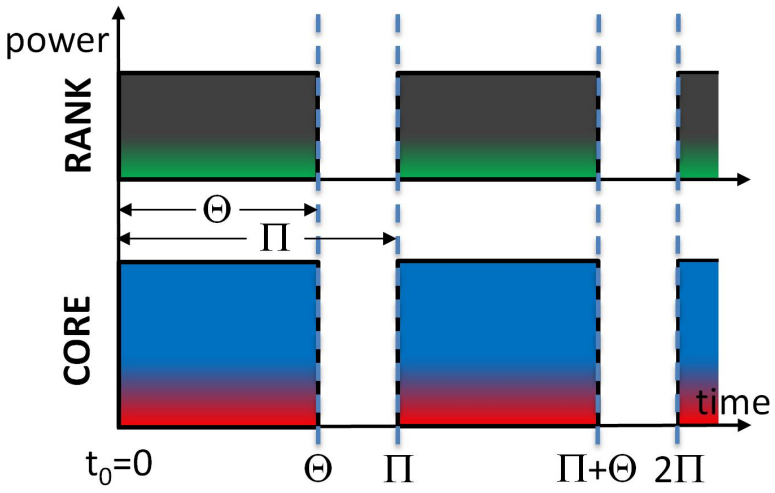


Figure 5.2: Periodic server time schedule example for any processing core P_k and its associated DRAM rank

To ensure deadlines of each sub-task-set assigned to each processing core in our system, we employ two approaches. The first schedulability condition was proposed by Shin and Lee [9], stated in Thm. 2.1 of chapter 2.

The second schedulability condition was proposed by Guo et.al. [80] when considering the harmonic periods of a task set, stated in the following theorem.

Theorem 5.1. *[9,80]. Given a single periodic resource server, with period Π_k, allocation time Θ_k and capacity $\mathbb{C}_k$, i.e. $\mathbb{C}_k = \Theta_k/\Pi_k$, and a task set Γ_k with its harmonic transformation Γ'_k with respect to Π_k, if $U(\Gamma'_k) \leq \Theta_k/\Pi_k$, then the task set Γ_k is schedulable on the periodic server under RM scheduling policy.*

We assume that each processing core will enter into a low-power mode (see chapter 2) during the time $\Pi_k - \Theta_k$. When the core exits from its low-power mode, it requires an extra overhead time Dl that must be accounted into the server allocation time Θ_k. Such Dl time is architecture dependent and considers delays such as CPU and DRAM power mode changes and caches cold starts.

In addition, we consider that each DRAM memory access has an specific delay time that depends on the accessing time to the caches, the DRAM commands decoding per-

formed by the MC and the DRAM data retrieving. Such a delay time is architecture dependent. Besides, we assume that each LLC miss is a DRAM memory access and that a local cache miss is stalling, which means whenever there is a miss in an LLC, the core is stalling until the cache-line is fetched from memory. Furthermore, we consider that preemption does not affect the number of cache-misses of a task (e.g. by partitioning cache to each task).

5.2.3 3D Platform Power Models

In this chapter we use the model for CPU as described in chapter 2, and to be conservative, the dynamic power switching activity factor parameter μ_i is defined based on its best-case bandwidth Z_BW_i, that is: $\mu_i = 1 - ((Z\_BW_i \cdot tDL)/(BxRQ))$, where tDL is the worst-case latency of a DRAM request, and $BxRQ$ is the number of bytes of each memory request. Thus, the CPU power consumed on a specific core and associated with each task τ_i can be formulated as Eq. 2.5, and for the sake of clarity we rewrite it as follows:

$$Pc_i = C0 + C1 \cdot \mathrm{T}_C(t) + \mu_i \cdot C2 \cdot (V_{dd}^{CPU})^3 \tag{5.1}$$

We also contemplate that the CPU has at least two power operation states (which may be applied to a wide range of today's microprocessors): (i) *Active (S1)*: Fully operational state; CPU is ON and operating at the maximum voltage and frequency; CPU is consuming both static and dynamic power, i.e. $P_{S1\_i}^{CPU} = P_{leak} + P_{dyn\_i}$. (ii) *Deep Sleep (S0)*: Stand-by mode state; CPU clock generator is OFF; Majority of CPU internal devices are OFF; delayed wake-up time; CPU is not consuming either static or dynamic power, i.e. $P_{S0}^{CPU} \cong 0$..

Also in this chapter we use the model for DRAM memory devices as described in chapter 2. As stated before, we assume that each bank of the same rank has the same bandwidth. Therefore, the bandwidth of each task τ_i in each bank is assumed to be

$H\_BW_{bank_i} = H\_BW_i \div (L-1)$. Also, since *active* and *read/write* powers are dependent on the bandwidth of τ_i, they can be simplified into a single constant P_{ARW} for the DRAM bank power calculation [157]. Consequently, the total power consumed by the DRAM bank when executing a task τ_i is as follows:

$$Pm_{bank_i} = P_{bg} + P_{ARW} \cdot H\_BW_{bank_i}. \tag{5.2}$$

Recall also from chapter 2 that DDR3 technology offers three different power-down modes to save power, and we are focused on the *self-refresh* power-down mode with power consumption denoted as P_{SR}^{DRAM}.

5.2.4 3D Platform Thermal Model

The 3D platform thermal behavior is similar to existing works (e.g. Zhou et.al. [190], Meng et.al. [128], and Chen et.al. [154]). 3D platforms have a relatively weak thermal correlation between banks or cores within the same layer. Instead, they have a much larger intra-layer thermal resistance than an inter-layer thermal resistance. Hence, it is reasonable to consider the thermal effect of one supercore (SC), consisting of a processing core plus its associated DRAM rank, to be isolated from the rest in the same platform, similar to [189, 190]. From this point, we will refer to each SC as SC_j as the combination of core P_j and its associated rank B_j.

Figure 5.3a shows our thermal model for one core with its associated rank of $L-1$ banks on top of it. Each bank in each layer is vertically interconnected by a series of thermal resistors $R_1, R_2, ..., R_L$ of small value in order to model the strong thermal correlation of vertically adjacent layers. The resistance R_c connecting the logic layer to the ambient includes the effect of the heat spreader and the heat sink. Each rank is consuming Pm_{bank} watts and the core is consuming Pc watts. The whole SC has a tight thermal coupling, modeled as a single thermal capacitance C_{sc}.

From the thermal model in Fig. 5.3a, a simplified thermal model is shown in Fig. 5.3b, with an equivalent power source and equivalent thermal resistance, P_{SC} and R_e respectively. The temperature in the only node of the equivalent model represents the temperature of the top DRAM layer of the SC. If such a top layer temperature T_{top} is kept under the temperature threshold, we can guarantee, in a stable state, that the temperatures for other layers beneath the top one are under the temperature threshold too. This conclusion is formulated in the following theorem.

Theorem 5.2. *Given a 3D platform with one logic layer closer to the system heat sink, and $L-1$ DRAM layers on top of it, modeled as an electrical circuit as shown in Fig. 5.3a, the peak temperature in the periodic stable state of any layer in the platform is lower than the peak temperature of the top DRAM layer.*

Proof. For the electrical circuit shown in Fig. 5.3a all the currents from the current sources (representing the power of each layer) are positive. In stable state, starting from the top node (layer) T_1, the current from the current source attached to each node must be added in the node, and the resulting current must flow to the node below through the inter-layer resistance. Therefore, the voltage of each inter-layer resistance has the positive polarity on its top node. Hence, since the temperature of each layer is represented by the voltage of each node in the network, the temperature of each layer will be larger or equal than the temperature of the layer beneath it. Thus, the farther a layer is from the heatsink, the hotter it may get. So, the temperature of the top layer must be larger than or equal than any other node in the network. □

The value of T_{top} depends on thermal resistances values and average power values Pm_{Γ_k} and Pc_{Γ_k} when a task set Γ_k is executed on a *SC* running a resource server of parameters $([\Pi_k, \Theta_k])$. We calculate $Pm_{\Gamma_k} = (\Pi_k/\Theta_k) \cdot \sum_{\tau_i \in \Gamma_k} U_i \cdot Pm_{bank_i}$ and $Pc_{\Gamma_k - dyn} = (\Pi_k/\Theta_k) \cdot \sum_{\tau_i \in \Gamma_k} U_i \cdot Pc_{i-dyn}$. Assuming $R_1 = R_2 = ... = R_L = R_m$, T_{top_k} can be calculated

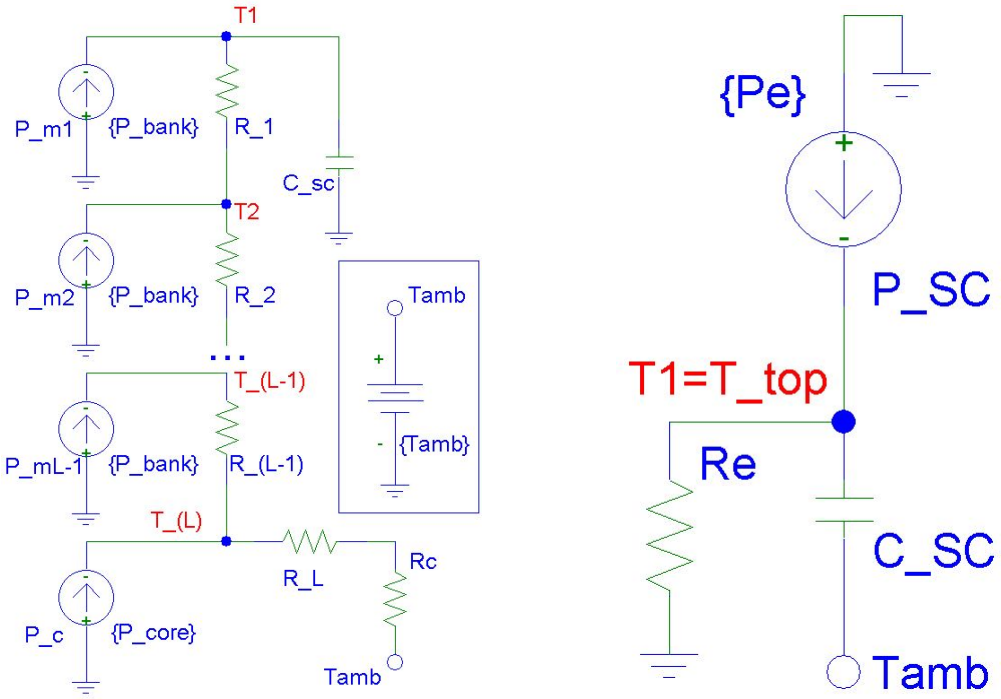


Figure 5.3: 3D Platform Super-Core Thermal Model and its Equivalent Circuit. (a) One core with its associated rank of $L-1$ banks on top of it. Each layer is vertically interconnected by thermal resistors $R_1, R_2, ..., R_L$. R_c includes the effect of the heat spreader and the heat sink. Each rank is consuming Pm_{bank} watts and each core is consuming Pc watts. *SC* tight thermal coupling modeled as a single thermal capacitance C_{sc}. (b) Equivalent thermal model with an equivalent power source and equivalent thermal resistance, P_{SC} and R_e respectively.

as:

$$T_{top\_k} = \frac{T_{ss} - ((T_{ss} - T_{amb}) \cdot (e^{-\Theta_k \cdot B})) - T_{amb} \cdot (e^{-\Pi_k \cdot B})}{(1 - e^{-\Pi_k \cdot B})}, \tag{5.3}$$

where, $T_{ss} = A/B$, with $A = (R_e \cdot C_{SC})^{-1} \cdot [\alpha \cdot Pm_{\Gamma_k} + \beta \cdot Pc_{\Gamma_k - dyn} + \beta \cdot C0 + T_{amb}]$ and $B = (R_e \cdot C_{SC})^{-1} \cdot [1 - \beta \cdot C1]$ [106]. Also, $\alpha = R_c \cdot (L-1) + R_m \cdot k_l$, and $\beta = R_c + R_m$, with $k_l = \sum_{x=1}^{L-1}(x+1)$, and $R_e = R_c + L \cdot R_m$.

5.2.5 Problem Formulation

Given the system models as introduced above, we can formally formulate our research problem as follows:

Problem 5.1. *Given a 3D platform with a logic layer of CMPs, multiple DRAM layers on top and a set of fixed priority real time tasks* Γ *scheduled using RMS, find the optimal task partition among the processing cores* $\{SC_1, SC_2, ..., SC_P\}$*, and the optimal parameters* $[\Pi_k, \Theta_k]$ *for each resource server implemented in each* SC_k *to guarantee the feasibility of the task set, i.e. the timing constraints of all tasks can be guaranteed while the peak temperatures in stable state of the processing cores and DRAM banks are kept under a given threshold (TempThr).*

5.3 Our Approach

To solve Problem 5.1, one key challenge is how to ensure that temperature for the processing core and memory does not exceed their threshold. While there have been extensive thermal aware techniques proposed on multi-core platforms (e.g. [155–157, 187]), most of them are responsive and best-effort in nature and cannot guarantee the temperature constraints. Temperature constraint violation can severely degrade memory performance and/or lead to unexpected processor shutdown, and thus make real-time tasks miss their deadlines. While there are some works (e.g. [194]) that analytically capture temperature variations based on existing power traces, how to optimize the task allocations in face of real-time and temperature constraints remains a problem. To solve this problem, we adopt the periodic resource model in our approach to manage the CPU and memory resources.

5.3.1 A Periodic Resource Model Based Approach

Different from the traditional resource model that keeps active when the ready queue is not empty, the periodic resource model proactively suspends the service for requests periodically but guarantees the availability of resources for Θ time units every Π time units. It has been well recognized [46] [80] that the periodic resource model can greatly

facilitate the analytical study of hierarchical resource sharing strategies with different scheduling algorithms for different services. Many feasibility conditions are introduced based on the periodic resource model.

Note that the periodic behavior of the periodic resource model also makes it a highly deterministic model for peak temperature guarantee based on Eq. (5.3). Given a task set Γ_k to be allocated to a supercore SC_k, and a specific server period value Π_k, we can readily develop an algorithm, as shown in Alg. 4, to determine (1) the allocation time for the periodic server (Θ_k), which can make the Γ_k feasible, and (2) the SC's peak temperature. The algorithm determines the two possible Θ_k using the two feasibility conditions presented in Theorems 2.1 and 5.1 (Lines 1 to 4), and the two corresponding peak temperatures are calculated using Eq. 5.3 (Line 5). If for the given task set Γ_k there exists a feasible Θ_k, then the algorithm returns the smallest possible peak temperature T_{peak} and the pair $[\Pi_k, \Theta_k]$ that allocated Γ_k (Line 7).

As shown in Alg. 4, for a given task set, we can readily determine its periodic server and analytically check its feasibility in terms of the timing and temperature constraints. The problem then becomes how to determine the task partitioning that can lead to the optimal solution. It is not difficult to see that the task partitioning results have profound impacts on parameter settings for periodic servers, and thus timing and temperature constraints. We next study how to develop appropriate task partitioning strategies.

5.3.2 Real-Time Task Partitioning Strategies

It is a well-known fact that task partitioning is a NP-hard problem [23]. Therefore, we focus on developing some effective and computationally efficient heuristics. To this end, we can readily design a resource constrained bin-packing based approach, called *Simple Combined Resource Usage Partitioning* (SCRUP) approach.

Algorithm 4 Function $FeasibilityChecking(\Gamma_k, \Pi_k)$

Input: $\Gamma_k = [C, T, H\_BW, Z\_BW]$, and Π_k (period for the periodic server).
Output: Peak temp. of top DRAM layer and periodic server config. $[\Theta_k, \Pi_k]$

1: Obtain the harmonic task set of Γ_k, i.e. Γ'_K, using Π_k as the base period;
2: **if** $U(\Gamma'_k) \leq 1$ **then** $\Theta_k^{HUB} = U(\Gamma'_k) \cdot \Pi_k$; **end if**
3: Let $N_k = |\Gamma_k|$ and $UB_{RM} = U(\Gamma_k)$; Using Eq. 2.1 solve for $\mathbb{C}_k$;
4: **if** $\mathbb{C}_k \leq 1$ **then** $\Theta_k^{SL} = \mathbb{C}_k \cdot \Pi_k$; **end if**
5: Use Eq. 5.3 to calculate $T^{hub}_{top\_\Gamma_k}$ for $[\Theta_k^{HUB}, \Pi_k]$ and $T^{sl}_{top\_\Gamma_k}$ for $[\Theta_k^{SL}, \Pi_k]$;
6: **if** $\exists$ feasible Θ_k **then**
7: Return $T_{peak} = \min_{[\Theta_k^{HUB}, \Pi_k] \wedge [\Theta_k^{SL}, \Pi_k]} \left(T^{hub}_{top\_\Gamma_k}, T^{sl}_{top\_\Gamma_k} \right)$, and $[\Theta_k, \Pi_k]$;
8: **else** Return $T_{peak} = \infty$, and $[0, 0]$; **end if**

Since resource utilization is closely related to the feasibility of a real-time task set, in the SCRUP approach, each task is characterized by *the Combined Resource Utilization* of processing core and memory. Specifically, the combined resource utilization $ResUtil_i$ of a task $\tau_i \in \Gamma$, is defined by the following equation.

$$ResUtil_i = U_i + U_i \cdot \left\lceil \frac{H\_BW_i}{BW^{max}} \right\rceil . \tag{5.4}$$

In Eq. (5.4), U_i is the CPU utilization required by task τ_i. Its memory bandwidth percentage H_BW_i/BW^{max} scaled by U_i indicates the fact that memory usage is needed only when τ_i is being executed. With the combined task utilization defined in Eq. (5.4), we can then sort the tasks based on their combined task utilization and then employ the traditional first-fit bin package method to allocate tasks to a SC. The detailed algorithm is shown in Alg. 5. Specifically, in Alg. 5, tasks are first sorted with their $ResUtil_i$ (Line 2). Subsequently, the algorithm packs tasks in the sorted order for the core verifying that the subtask set chosen is feasible (*for loop* lines 8 to 12). Such a subtask set is assigned to the next available core and the process is repeated over the remaining unallocated tasks until there are no more tasks to allocate or no more available cores (*while loop* line 5). The algorithm returns if the task set is feasible or not. If feasible, it also returns the sub task set allocated to each core.

While the combined resource utilization helps to identify the resource requirement of different sources by a task, one major pitfall of this metric is that two tasks with the same combined resource utilization may have different overall power consumption. In addition, even though both tasks may consume the same amount of total power by CPU and memory, the result peak temperatures may be different as indicated in Eq. (5.3). Therefore, to better deal with the temperature constraints, we develop another metric, i.e. *the Nominal Total Power consumption* (NTP), to measure the combined resource usage, which is defined in the following equation:

$$NTP_i = (\beta \cdot U_i \cdot Pc_{i-dyn}) + (\alpha \cdot U_i \cdot Pm_{bank_i}), \tag{5.5}$$

where Pc_{i-dyn} and Pm_{bank_i} are power consumptions for processing core (dynamic power) and memory when executing task τ_i, which are defined in Eq. 2.5 and 2.7, respectively. α and β are two constants defined in Eq. (5.3). Essentially, NTP_i is the equivalent total power consumption when a task is executed in the reduced thermal model as depicted in Figure 5.3b.

With the introduction of a new metric as shown in Eq. (5.5), we can develop another algorithm, called *the Thermal Aware Task Partitioning* (TATP) approach. The TATP approach follows the same algorithm structure as that in Alg. 5. The only difference is that, instead of sorting tasks based on their combined resource utilization as defined in Eq. (5.4), the TATP approach sorts tasks based on the nominal total power consumption of each task as defined in Eq. (5.5). The approach is also shown in Alg. 5, considering the change of line 2 by line 3 and performing the new sorting in line 6.

The previous two approaches partition tasks based solely on their resource utilization, and the feasibility and periodic server settings are determined based on Alg. 4. It is a well-known fact that the relationship of task periods, if explored appropriately, can significantly increase the feasibility [75]. For example, a harmonic task set can be schedulable using

Algorithm 5 Approach SCRUP **OR** TATP

Input: $\Gamma = [C, T, H\_BW, Z\_BW]$, $TempThr_{SC}$, $\mathcal{SC} = \{SC_1, ..., SC_P\}$, L, Π_{test}.
Output: Whether Γ is feasible or not, and allocation for each $\tau_i \in \Gamma$ to a SC_j

1: **for all** $\tau_i \in \Gamma$ **do**
2: Let $ResUtil_i = U_i + U_i \cdot \left\lceil \frac{H\_BW_i}{BW^{max}} \right\rceil$, using Eq. 5.4, **OR**
3: Let $NTP_i = (\beta \cdot U_i \cdot Pc_i) + (\alpha \cdot U_i \cdot Pm_{bank_i})$, using Eq. 5.5
4: **end for**
5: **while** $\Gamma \neq \emptyset$ && $|SC| \neq 0$ **do**
6: Sort tasks $\tau_i \in \Gamma$ according to $ResUtil_i$ **OR** NTP_i in descending order;
7: $n = |\Gamma|$; $\Gamma_k = \emptyset$;
8: **for** $i = 1$ to n **do**
9: $\Gamma_k = \Gamma_k + \tau_i$;
10: $T_{peak} \leftarrow FeasibilityChecking(\Gamma_k, \Pi_{test})$;
11: **if** $T_{peak} > TempThr_{SC}$ **then** $\Gamma_k = \Gamma_k - \tau_i$ **end if**
12: **end for**
13: Assign Γ_k *to* $SC_k \in \mathcal{SC}$; $\mathcal{SC} = \mathcal{SC} - SC_k$; $\Gamma = \Gamma - \Gamma_k$;
14: **end while**
15: **if** $\Gamma \neq \emptyset$ **then** Return: Γ is not schedulable; **end if**

RMS with total utilization reaching as high as 1. Even for tasks that are not perfectly harmonic, a *harmonic index* has been developed [172] to quantify the harmonicity among task sets to improve the feasibility of task partitioning results. Moreover, as shown in the existing work [80], making the period of the periodic server harmonic to the task set can greatly reduce the resource usage. Based on the existing work, we can readily prove the following theorem.

Theorem 5.3. *When scheduling a task set, i.e.* Γ *, with total utilization* U_Γ *on a periodic server (*Θ*,* Π*), the capacity of the periodic server, i.e.* $\mathbb{C} = \Theta/\Pi$*, is minimized if the task set is perfectly harmonic and the server period is also harmonic to the task set.*

Proof. The reader may recall that the minimum server allocation time Θ to allocate a task set Γ (with total utilization U_Γ) using a server with period Π may be obtained using two methods as explained in section 5.2. According to Thm. 5.1, if Γ is a perfectly harmonic task set and it is also perfectly harmonic with respect to Π, then the minimum server capacity to allocate Γ is equal to the total task set utilization U_Γ. On the other hand, it is

possible to find the minimum server capacity to allocate any task set Γ according to Thm. 2.2 from chapter 2. Thus, we need to proved that:

$$U_\Gamma \overset{?}{\leq} \frac{U_\Gamma}{log\left\lceil \frac{2k+2(1-U_\Gamma)}{k+2(1-U_\Gamma)} \right\rceil} = AB_{RMS}, \tag{5.6}$$

for any valid k and U_Γ. Thus, we may have:

$$\begin{aligned} log\left\lceil \frac{2k+2(1-U_\Gamma)}{k+2(1-U_\Gamma)} \right\rceil &\overset{?}{\leq} 1. \\ \left\lceil \frac{2k+2(1-U_\Gamma)}{k+2(1-U_\Gamma)} \right\rceil &\overset{?}{\leq} 10, \\ 2k+2(1-U_\Gamma) &\overset{?}{\leq} 10k+20(1-U_\Gamma), \\ 0 &\overset{?}{\leq} 8k+18(1-U_\Gamma). \end{aligned} \tag{5.7}$$

Since, k must be an positive integer and U_Γ must be a positive number between 0 and 1, therefore the inequality in 5.6 is true. □

Theorem 5.3 implies that a task set Γ_k with the same utilization may need a periodic server with smaller server capacity. Also, choosing the server period harmonic with the task set also helps to minimize the server capacity. Since a small server capacity can lead to a lower peak temperature, we develop another task partitioning approach, called *Harmonic and Temperature Aware Task Partitioning* (HTTP) approach, able to take advantage of the harmonic relationship among task periods to improve the task partitioning results.

Our HTTP approach is formulated in Alg. 6. First, tasks are sorted by their periods T_i (line 3). Then, using $\tau_i \in \Gamma$ as a reference task, the algorithm sorts all other tasks based on the harmonic index related to the reference task (lines 6-7). Next, it picks a timing feasible sub task set with the most harmonic tasks with respect to the reference task, such that the total transformed utilization is maximum, and the SC's peak temperature is below the threshold (lines 8-10). Among all possible n sub task sets, the one with the highest

Algorithm 6 Harmonic and Temp. Aware Task Part. (HTTP)

Input: $\Gamma = [C, T, H\_BW, Z\_BW]$, $TempThr_{SC}$, $\mathcal{SC} = \{SC_1, ..., SC_P\}$, L, Π_{test}.
Output: Whether Γ is feasible or not, and allocation for each $\tau_i \in \Gamma$ to a SC_j

1: $\Gamma_{TS} = \emptyset$;
2: **while** $\Gamma \neq \emptyset$ && $|SC| \neq 0$ **do**
3: Sort τ_i increasing order with respect to T_i;
4: $n = |\Gamma|$; $U_{TS} = -\infty$;
5: **for** $i = 1$ to n **do**
6: Construct Γ' (Sub-Harmonic of Γ) using DCT [75] with τ_i as base;
7: Sort all $\tau_j \in \Gamma$ in increasing order w/ respect to $H(\tau_j) = \frac{U'_j - U_j}{U_j}$;
8: Γ_{k_j} = pick up k_j tasks from Γ s.t. for the corresponding $\Gamma'_{k_j} \in \Gamma'$:
9: (1) $U(\Gamma'_{k_j}) \leq 1$; **AND** (2) $|\Gamma'_{k_j}|$ is maximized; **AND**
10: (3) $T_{peak} \leq TempThr_{SC}$ ($T_{peak} \leftarrow FeasibilityChecking(\Gamma_{k_j}, \Pi_{test})$)
11: **if** Γ'_{k_j} is feasible **AND** $U(\Gamma_{k_j}) > U(\Gamma_{TS})$ **then** $\Gamma_{TS} = \Gamma_{k_j}$;
12: **end for**
13: Assign Γ_{TS} *to* $SC_k \in \mathcal{SC}$; $\mathcal{SC} = \mathcal{SC} - SC_k$; $\Gamma = \Gamma - \Gamma_{TS}$;
14: **end while**
15: **if** $\Gamma \neq \emptyset$ **then** Return: Γ is not schedulable; **end if**

total utilization (Line 11) is allocated to an SC (line 13). The remaining tasks will go through the same procedure again until there are no more tasks to allocate or there are no more available cores (*while loop* line 2). The algorithm returns if the task set is feasible or not. If feasible, it also returns the sub task set allocated to each core.

5.4 Experiments, Analysis and Results

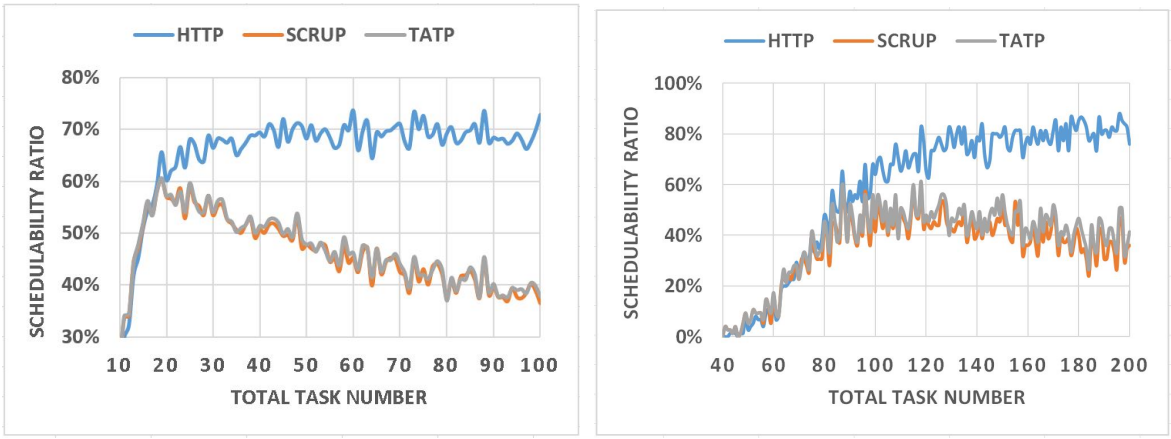
In section 5.3.2 different approaches are proposed. It is hard to prove if one dominates the other analytically. Therefore, we use simulation results to study their performance and compare them.

In our simulation, we adopted the similar 3D platform parameters used in previous works, such as [59, 128, 154]. We target architectures consisting of a single logic layer closer to a heat sink with 4 and 16 symmetrical and homogeneous cores ($SC = \{4, 16\}$), plus four DRAM memory layers ($L = 5$) on top of the logic layer. Thus, we have four

banks per each rank symmetrically stacked on its associated core. We adopted the power model for the DDR3 DRAM chip of 1GB, using a low conductivity substrate, for each DRAM layer. Hence, each layer has 1 GB and 4GB in total with four layers. Each bank has 64 MB and each rank has 256 MB. We assume that each core has an in-order execution architecture and power model constants similar to [106]. We consider the CPU and DRAM subsystems running at the same frequency of 2000 MHz. We also consider that each DRAM rank drives a 1024 TSVs bus with a DRAM best-case delay of $30ns$ ($4ns$ LLC, $24ns$ MC, $2ns$ DRAM) [54]. The values for constants P_{bg} and P_{ARW} are very similar to the ones reported in [157]. Based on this model, we observe that the DRAM power is comparable to the core power, which fits the power ratio of core and memory reported in the literature [59]. We assume an ambient temperature value of $35°C$.

We randomly generated 12000 task sets for our simulation for each architecture and for each temperature threshold tested. Specifically, the utilization for each task is generated randomly using the UUnifast algorithm [195], which provides a uniform distribution for all task utilizations while keeping fixed the total task set utilization. For the case of 4 cores architecture, the total task set utilization falls in the range $U(\Gamma) = [0.49, 0.57]$ for $TempThr_{SC} = 70°C$ and $U(\Gamma) = [0.56, 0.65]$ for $TempThr_{SC} = 75°C$, and for 16 cores the range is $U(\Gamma) = [2.08, 2.25]$ for $TempThr_{SC} = 70°C$ and $U(\Gamma) = [2.36, 2.57]$ for $TempThr_{SC} = 75°C$. Likewise, the maximum tasks bandwidth is randomly generated in the range $H\_BW_i = [17, 34]GB/s$, and the minimum $Z\_BW_i = [0.34, H\_BW_i]GB/s$. For overhead, we choose a value of 0.5 ms which is able to account for CPU and memory wake ups, plus cache cold starts. Since a smaller server period offers a smaller SC's peak temperature [194], we choose the value of $\Pi_{test} = 10\ ms$.

Fig. 5.4 and 5.5 show the schedulability ratio of the task sets versus a fixed number of tasks (x-axis), when setting $TempThr_{SC} = 70°C$ and $TempThr_{SC} = 75°C$, respectively. Fig. 5.4a and 5.5a for an architecture of 4 cores, and Figures 5.4b and 5.5b for an archi-

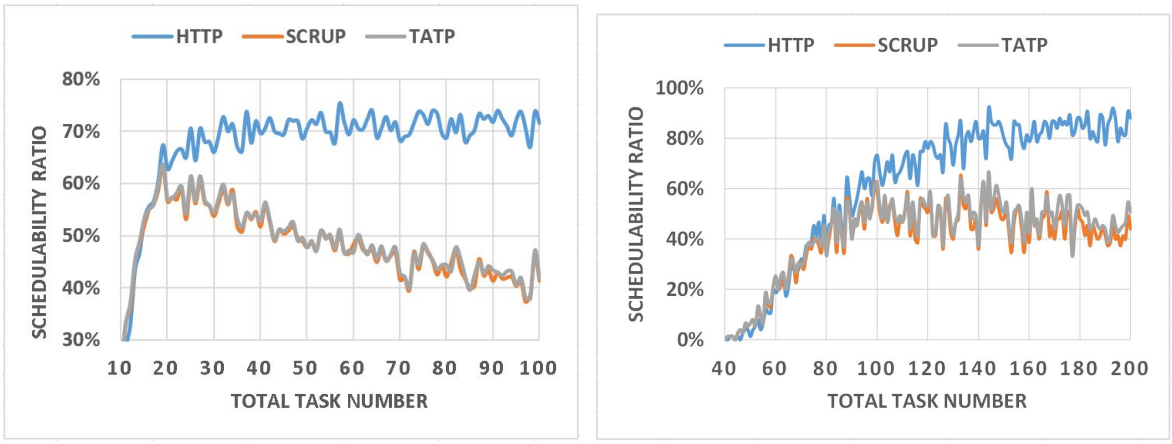


(a) (b)

Figure 5.4: Task set feasibility comparison by total number of tasks in Γ with $TempThr_{SC} = 70°C$. (a) For 4 *SC* and (b) For 16 *SC*.

tecture of 16 cores. Both Fig. 5.4 and 5.5 show a significant increment in schedulability ratio of HTTP compared to the classic scheduling methodologies SCRUB and TATP, for both analyzed architectures. The schedulability of task sets with a small number of tasks is reduced because some tasks may have average power values that prevent them from being allocated alone to any SC. Also, it can be seen in Fig. 5.4a and 5.5a that SCRUB and TATP decreases with a larger number of tasks. This effect is due to the fact that with a smaller number of tasks, we may have a higher harmonicity value. Thus, the server capacity to guarantee the timing of the sub task set is usually calculated using the schedulability condition of Thm. 5.1, which tends to be lower than the server capacity calculated using Thm. 2.1. A smaller server capacity leads to a smaller SC's peak temperature. Therefore, it is possible to allocate more tasks to the same core keeping the peak temperature under the temperature threshold.

Table 5.1 shows the total schedulability ratio per method with the two tested temperature thresholds. A 19.5% average increment of HTTP over SCRUB and TATP can be obtained from the data. This is because HTTP allocates sub-task sets with a higher harmonicity value, which is directly translated into a smaller allocation time Θ_k, which



(a) (b)

Figure 5.5: Task set feasibility comparison by total number of tasks in Γ with $TempThr_{SC} = 75°C$. (a) For 4 *SC* and (b) For 16 *SC*.

Table 5.1: Schedulability ratio per method with two temperature thresholds

SCs	$TempThr_{SC}$	HTTP	SCRUP	TATP	TATP–SCRUB
4	$75°C$	67.84%	48.27%	48.65%	0.38%
	$70°C$	65.91%	46.14%	46.66%	0.52%
16	$75°C$	61.20%	40.05%	42.11%	2.06%
	$70°C$	58.63%	35.43%	38.07%	2.64%

derives in a smaller server capacity, and, therefore, a smaller SC's peak temperature. It is noteworthy to mention that with a tighter temperature, TATP increases the schedulability ratio compared to SCRUB (see the last column of table 5.1), because TATP considers the thermal and power parameters of the system, which become more significant than the actual utilization of each task with a lower temperature threshold.

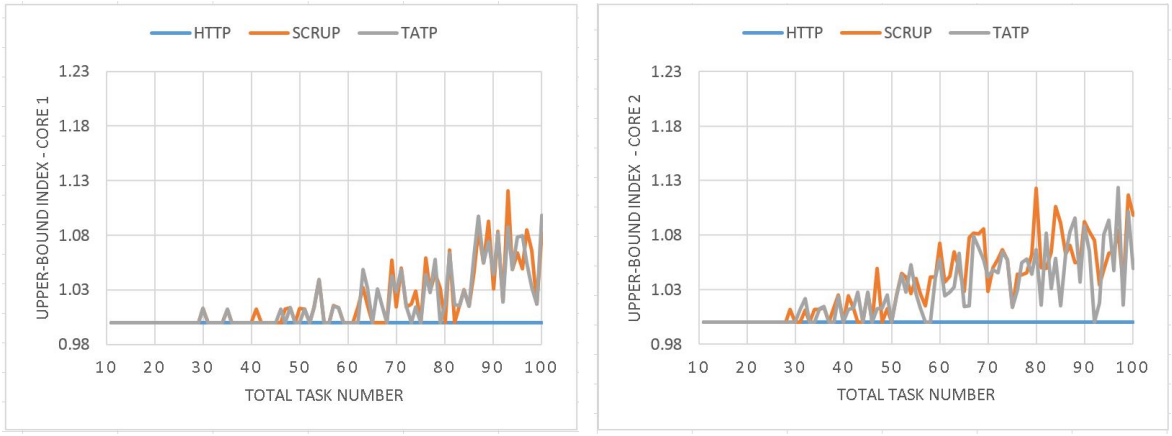
As shown in Alg. 4 there are two different methods to obtain the server allocation time (Θ_k) for an specific server period (Π_k),i.e., either using the Harmonic-Upper Bound method to obtain Θ_k^{HUB} (using Thm. 5.1) or using the ShinLee method to obtain Θ_k^{SL} (using Thm. 2.1), whichever is smaller. For the next set of experiments we want to test how many task sets were schedulable with using either of the two methods. Hence, we will use a new metric called the *Upper-Bound Index* (UBI) value per task number. First,

let each schedulable sub-task set assigned to each SC to have a tag ($V_{\Gamma_{SC}}$) of either 1 or 2. If the sub-task set is found schedulable obtaining Θ_k^{HUB}, the tag of such a sub-task set is assign to 1 ($V_{\Gamma_{SC}} = 1$). On the other hand, if the task set is found schedulable obtaining Θ_k^{SL}, the tag of such a task set is assign to 2 ($V_{\Gamma_{SC}} = 2$). Thus, the $UBI(n)$ value per task number n is equal to the average value of all the V_Γ of task sets with the same specific task number n. Fig. 5.6 and Fig. 5.7 plot the UBI values for each integer value of $n = [10, 100]$ and for each core in an architecture of four cores when $TempThr_{SC} = 70°C$ and $TempThr_{SC} = 75°C$, respectively.

By taking a closer look to each method to obtain the minimum server allocation time, we can see that for the same Π_k, Θ_k^{HUB} tends to be smaller than Θ_k^{SL}, which directly translates into a smaller server capacity and therefore a smaller peak temperature for both the CPU and the memory layers. Of course, this condition is not always true and depends on the relationship of the task periods of the tasks assigned on each sub-task set. Thus, UBI(n) is an indicator of how much an scheduling method (HTTP for example) is able to partition tasks such that actually $\Theta_k^{HUB} < \Theta_k^{SL}$. Hence, it can be seen in Fig. 5.6a and Fig. 5.7a that HTTP is able to find the server allocation time for all the sub-task sets in the first core using always the Harmonic-Upper Bound method, because $\forall n, UBI(n) = 1$. Also, the UBI(n) values using HTTP for cores 2 and 3 (Fig. 5.6b, 5.6c, 5.7b and 5.7c) are equal to 1 for almost all n values. Plus, the UBI(n) values using HTTP for core 4 (Fig. 5.6d and 5.7d) are smaller than the values obtained with TATP and SCRUB.

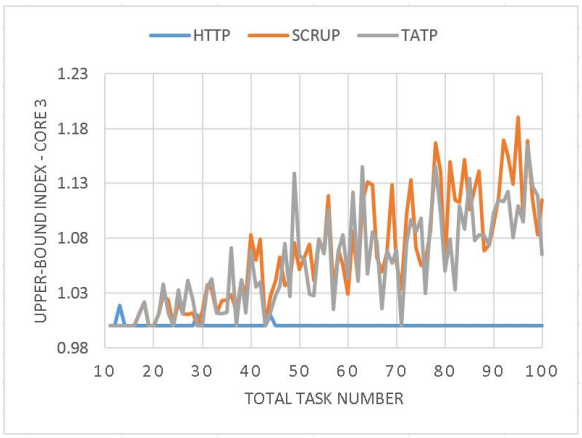
This results demonstrate that by allocating tasks with a higher harmonicity on the same sub-task set (by using the HTTP method), it is possible to obtain a smaller server allocation time for the same server period value, and thus obtain a smaller SC peak temperature, which translates directly into an increase in schedulability ratio as seen in Fig. 5.4 and 5.5.

Figure 5.6: Upper-Bound Index by total number of tasks in Γ with $TempThr_{SC} = 70°C$ and $SC = 4$. (a) For $Core1$, (b) For $Core2$, (c) For $Core3$, (d) For $Core4$.

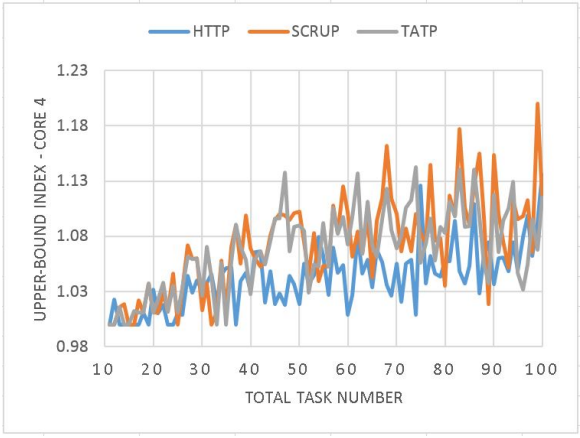


(a)

(b)



(c)



(d)

Figure 5.7: Upper-Bound Index by total number of tasks in Γ with $TempThr_{SC} = 75°C$ and $SC = 4$. (a) For $Core1$, (b) For $Core2$, (c) For $Core3$, (d) For $Core4$.

5.5 Summary

The increased power density and low heat dissipation efficiency have presented a great challenge on how to develop real-time applications on 3D IC platforms. As memory power consumption continues to grow to be comparable or even exceed the core power consumption, we believe that it is critical to schedule both processing cores and memory in a collaborative manner to deal with the thermal challenges. We develop a novel strategy that employs periodic resource servers to ensure timing and thermal constraints for real-time tasks on a 3D platform. Specifically, we take considerations of different factors, such as core and memory resource requirement, power consumption, the harmonic relationship among tasks, and timing and temperature constraints in our approach to partition real-time tasks on different cores. Simulation studies show that our proposed method can schedule on average 19.5% more tasks than the comparative methodology based on previous allocation mechanisms.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

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

6.1 Summary

Many computing platforms and especially embedded systems are being developed with real-time capabilities by multiple industries, such as automotive, biomedical and aerospace, covering a spectrum from the very simple to the very complex, making real-time systems ubiquitous and critical to our personal and social life. A real-time system is a system whose execution time is expected to comply with deadlines, and missing a deadline is as negative as the incorrect output from the computation, leading to catastrophic consequences for certain applications. As a result, the most important requirement of a real-time system is predictability and not performance, which makes the correct implementation of a system scheduler to ensure timing constrains very important. Traditionally, when designing scheduling strategies, designers have considered only CPU and performance characteristics of the system. However, real-time systems processing huge streaming of data, such as cameras and specialized sensors, are becoming popular. Such applications not only generate large amounts of I/O workloads, but also become more and more memory intensive, which is translated in the need to develop real-time platforms with higher memory bandwidth and capacity. Hence, the restrictions and latencies imposed by memory devices have gained a significant impact, not only on the execution time, but also on other aspects, such as power consumption and temperature of operation. Since memory devices are expected to keep increasing in size, capacity, bandwidth and power consumption, we believe that today and future real-time system scheduler mechanisms must be

developed considering not only the characteristics of the processing units of the system, but also the increasing restrictions and latencies imposed by memory devices. In this dissertation, we analyzed two memory-related scheduling problems affecting the correct design of a real-time system.

First, real-time systems encounter an important degradation of predictability due to the uncertainties in the execution time imposed by an unbounded number of cache memory accesses. This is aggravated for real-time systems developed on multi-core platforms with a common shared cache memory across a group of cores. We studied how to enhance the predictability by partitioning cache memory on a multi-core platform when scheduling hard real-time tasks, and we developed two static schemes for cache allocation and task partitioning that consider not only the execution time variations with cache allocations but also the task period relationship, showing a significant improvement in the feasibility of the system. Both of them are able to significantly improve the system resource usage and the schedulability of hard real-time tasks, when compared with other scheduling mechanisms. From our experimental studies, we can see that our second approach is able to increase the schedulability of hard real-time tasks by up to four times, when compared to conventional partitioned RMS.

Second, since the power consumption of computing systems has been increasing exponentially, resulting in tremendous heat generation, many thermal management techniques have been proposed. Most of them focus exclusively on either CPU or memory. Moreover, most of such techniques are on-line reactive in nature, which threatens the predictability of real-time systems. In this dissertation, we studied the problem of how to guarantee timing constraints for hard real-time systems under CPU and memory thermal constraints from two perspectives. On one hand, for the case of a single core and its main memory individually packaged, but with a high thermal interaction. We developed a novel strategy that employs the periodic server model to collaboratively schedule both the CPU

and memory to meet the timing constraints for a hard real-time task set, and in the meantime satisfy the temperature constraints for both the CPU and memory devices. From our experimental studies, we can demonstrate the effectiveness of our proposed approach in reducing the peak temperature, as well as the need to take both the CPU and memory systems into consideration simultaneously for system-level thermal management. On the other hand, we extended our analysis for the case of multi-core architectures where both CPU and memory devices are combined into a single package in a 3D integrated platform. Our proposed scheduling method that employs periodic resource servers is able to effectively guarantee the thermal constraints of both the logic layer and memory layers, as well as the timing constrains of hard real-time tasks when deployed using a 3D IC integrated platform. From our experimental studies, we can conclude that our proposed method can schedule on average 19.5% more tasks than the comparative methodology based on previous allocation mechanisms.

6.2 Future Work

In this dissertation, we primarily focus on developing scheduling mechanisms for hard real-time tasks when considering restrictions imposed by memory devices, including restrictions in temperature of operation. In particular, we can extend our research to include and test additional aspects or parameters of different platforms and real-time systems in our proposed scheduling mechanisms. First, our set of scheduling mechanisms proposed in chapter 3 can be extended to partition not only cache memories but also DRAM memories using bank partitioning [196]. The interaction between both methodologies of cache and bank partitioning when maximizing the schedulability of the system can be further explored. Next, our scheduling mechanism proposed in chapter 4 can be added with an additional abstraction layer by including multiple periodic servers capable of scheduling

different types of tasks. The problem becomes how to characterize the tasks in the system in order to allocate them in the different servers, keeping the goal of maximizing the schedulability of the system while keeping the peak temperature of both the CPU and memory under threshold. Then, another problem is how to assign different resources to each server in terms of allocation time, processor speed and memory bandwidth. Finally, our scheduling mechanism proposed in chapter 5 can also be extended by relaxing the constraints of the system model. For instance, since it is known that the performance of each task is not a linear function with respect to the memory bandwidth, it is possible to allow that each rank in the system is managed by the scheduler depending on the requirements of the server by implementing a memory throttling mechanism [63]. We believe this would further reduce the power consumed by the memory on some of the supercores, allowing a larger number of tasks to be allocated in the system.

Also, as mentioned in chapter 5, 3D Integrated Circuits (3D-ICs) is a promising solution to overcome the Processor-Memory Gap problem because it is possible to include a high-capacity memory with a high bandwidth. However, the current trend in chip manufacturing adopted by the research community and the industry regarding the 3D-ICs architectures is changing rapidly. Therefore, we believe that it is necessary to consider such memory architectures and technologies when designing future schedulers to guarantee hard-real time tasks' timing. With this in mind, in the following paragraphs we elaborate more on this idea.

The semiconductor industry is actively pursuing 3D-ICs with Through-Silicon Via (TSV) technology, which allows the integration of two different layers of chips with different manufacturing processes into a single package one of top of the other, reducing the wires' size and the communication path. However, while the micron-ranged TSV pitches are enough for stacking memory on top of processors and memory-on-memory stacking, they may not be enough to significantly mitigate the well-known on-chip interconnect

problems. Hence, a new manufacturing process called monolithic 3D-ICs offers through-silicon connections with 50*nm* diameter or less and is capable to provide 10,000 times the areal density of TSV technology [197].

Qualcom is producing or planning to produce their 3D-ICs for mobile applications using monolithic 3D technology, which they claim allows them to use edge manufacturing technology of 10 nm or less, in the first layer and 28 nm for the top layers. Also, according to their floorplanning methodology, they can achieve temperatures comparable or even smaller than the ones obtained with 2D technology [198], by using 2D floorplanning for creating 3D integrated designs using monolithic 3D [199]. Their process can achieve up to 12% and 8% power savings for a single block and SoC, respectively, when compared with their 2-D counterparts implemented using commercial tools. According to them, it is possible to achieve a reduction in power consumption in a 3D platform but still the thermal problem in the platform exists.

Monolithic 3D ICs bring new improvements compared to other 3D integration technologies, but they require a different analysis in terms of power and thermal management. For instance, authors in [200] focus on thermal-aware 3D floorplanning for logic-to-logic integration for microprocessors targeting mobile applications, because monolithic 3D reduces the power of the chips, but still the power density is high, increasing the temperature of the whole chip. The major bottleneck of considering thermal aspect within the physical design process is the huge runtime required for accurate temperature analysis. Therefore, the authors of this paper identify the factors affecting temperature and develop a very fast and accurate non-linear regression-based temperature evaluation model for monolithic 3D ICs. Also, they show that monolithic 3D ICs have almost zero lateral conduction at the source of power due to very thin layers and show no lateral spreading in the device layers, developing a methodology to obtain packaging-aware fast and accurate thermal analysis models for monolithic 3D ICs with a different number of stacking layers. They

used this model in a thermal-aware floorplanner to show significant temperature reduction with minimum area overhead. The speed of their thermal model enables them to use it in the floorplanning process without any runtime issues, and faster than simulators such as HotSpot and its patch extension for 3D thermal modeling. Depending on the design power density, 5% to 16% total power reduction in 3D-ICs is needed before junction temperature can be similar to that of the corresponding 2D-ICs [201]. Authors conclude that emerging monolithic 3D-ICs integration technologies are actually thermally friendly for mobile applications due to their corresponding boundary conditions, as well as their relatively lower total power budget.

Some works started proposing the combined manufacturing of memory and logic devices using monolithic 3D approaches [202]. The problem is that common fabrication technologies for DRAM layers (NMOS) are not the same as for logic layers (CMOS/FFET). Thus, it is necessary to generate different mechanisms for manufacturing and different architectures to include memory devices into the 3D chip, which brings the need to account for different thermal models that include the new memory technologies.

Generally speaking, many research works are aimed to develop mechanisms to perform thermal management on 3D platforms but with logic-to-logic integration. One reason for this is because the integration of memory devices is difficult due to the alignment problems at manufacturing and the big size of the TSVs to communicate the different layers. Moreover, the interposers and bonding materials are much thicker, making the heat dissipation a problem. However, using monolithic manufacturing, the layers are build on of top of the other with a very thin separation between both, making the thermal dependencies practically between inter-layer thermal resistances only.

Equally important is the fact that in the future with the power increasing and the dark silicon becoming more dominant in some devices, adaptive multi-core architectures may improve the performance of some systems. The idea is to dynamically use only the pro-

cessor parts needed, including on-chip memories, such as caches, scratch pad memories, DRAM caches, and 3D on-chip DRAMs. For instance, research works such as [203] are working specifically on reconfigurable hardware and the best way to adapt it according to the current running application. However, there still remains the challenge of how to co-schedule the CPU with the memory devices in such architectures.

In addition, chip manufacturers are still paying close attention to the 2.5D integration process because it is cheaper and is readily available for commercial devices. 2.5D is now seen as a way to improve yield by manufacturing smaller parts/chips, then connecting them on a single interposer. It would be possible to implement and manufacture separately, in a single many-core platform, different types of memory devices, including the cache, DRAM cache, or even on-chip DRAM [204].

We believe that the problem of how to perform the schedule of computing applications including real-time ones still remains open, considering the new memory layouts and features or how to perform the co-scheduling between CPU and memory using the above-mentioned architectures. Also, another research problem that still remains open is to how to increment the performance of memory-bounded applications, while implementing an efficient thermal management using a diversified memory hierarchy with different memory technologies on different layers, perhaps with some memories on-chip when using a monolithic 3D-IC, or a combination of 3D-ICs inter-communicated using a 2.5D integration architecture. As an illustration, the new generation of Xeon processors from Intel (KNL) [205] are able to provide on-chip 3D DRAM, configurable in the machine BIOS as cache, addressable memory or a mixture, depending on the application. In the future, some platforms may even offer the possibility to change the memory configuration online. If it could be the case, what would be the optimal configuration for memory to maximize performance, power, energy, reliability, and temperature of operation among other optimization parameters when scheduling real-time systems?

BIBLIOGRAPHY

[1] 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), 2011 IFIP 9th International Conference on*. IEEE, 2011, pp. 27–32.

[2] C.-W. Chang, J.-J. Chen, T.-W. Kuo, and H. Falk, "Real-time task scheduling on island-based multi-core platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 2, pp. 538–550, 2015.

[3] U. Tech., "2015 embedded market study. Then, now: What's next?" UBM, Tech. Rep., 2015.

[4] J. Cooke, "IoT, wearable, networking and automotive markets driving external memory innovation," 2016. [Online]. Available: http://www.memcon.com/pdfs/proceedings2016/KEY103.pdf

[5] A. Shilov, "Price check: Price gap between DDR3 and DDR4 memory almost gone," 2016. [Online]. Available: https://www.anandtech.com/show/10058/price-check-price-gap-between-ddr3-and-ddr4-memory-almost-gone

[6] T. Kelter, "WCET Analysis and Optimization for Multi-Core Real-Time Systems," Ph.D. dissertation, Technischen Universitat Dortmund, 2015.

[7] K. Tran and J. Ahn, "HBM: Memory solution for high performance processors," *Proceedings of MEMCON*, 2014.

[8] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu, "Tiered-latency DRAM: A low latency and low cost DRAM architecture," in *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*. IEEE, 2013, pp. 615–626.

[9] I. Shin and I. Lee, "Compositional real-time scheduling framework with periodic model," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 3, p. 30, 2008.

[10] G. Fagas, J. P. Gallagher, L. Gammaitoni, and D. J. Paul, "Energy Challenges for ICT," in *ICT-Energy Concepts for Energy Efficiency and Sustainability*. InTech, 2017.

[11] K. Rupp, "40 years of microprocessor trend data," 2015. [Online]. Available: https://www.karlrupp.net/2015/06/40-years-of-microprocessor-trend-data/

[12] G. Rose, "Alleviating memory and cache contention in safety-critical applications," 2014. [Online]. Available: https://www.aerodefensetech.com/component/content/article/adt/features/articles/20339#

[13] B. Jacob, S. Ng, and D. Wang, *Memory systems: cache, DRAM, disk.* Morgan Kaufmann, 2010.

[14] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications.* Springer Science & Business Media, 2011, vol. 24.

[15] M. Qiu and J. Li, *Real-time Embedded Systems: Optimization, Synthesis, and Networking*. CRC Press, 2011.

[16] A. Bastoni, B. Brandenburg, and J. Anderson, "Cache-related preemption and migration delays: Empirical approximation and impact on schedulability," *Proceedings of OSPERT*, pp. 33–44, 2010.

[17] S. Altmeyer, R. I. Davis, and C. Maiza, "Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems," *Real-Time Systems*, vol. 48, no. 5, pp. 499–526, 2012.

[18] S. Mittal, "A survey of architectural techniques for dram power management," *International Journal of High Performance Systems Architecture*, vol. 4, no. 2, pp. 110–119, 2012.

[19] J. John, "Global Embedded Systems Market Will Reach USD 225.34 billion by 2021," June 2017, [Online; posted 8-June-2017]. [Online]. Available: https://globenewswire.com/news-release/2017/06/08/1010414/0/en/Global-Embedded-Systems-Market-Will-Reach-USD-225-34-billion-by-2021.html

[20] E. A. Lee and S. A. Seshia, *Introduction to embedded systems: A cyber-physical systems approach.* MIT Press, 2016.

[21] A. Mohammadi and S. G. Akl, "Scheduling algorithms for real-time systems," *School of Computing Queens University, Tech. Rep*, 2005.

[22] R. Mall, *Real-time systems: theory and practice.* Pearson Education India, 2009.

[23] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM computing surveys (CSUR)*, vol. 43, no. 4, p. 35, 2011.

[24] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[25] J. W. Liu, *Real-time systems*. Prentice Hall PTR, 2000.

[26] M. A. Hoque, M. Siekkinen, K. N. Khan, Y. Xiao, and S. Tarkoma, "Modeling, profiling, and debugging the energy consumption of mobile devices," *ACM Computing Surveys (CSUR)*, vol. 48, no. 3, p. 39, 2016.

[27] G. Singla, G. Kaur, A. K. Unver, and U. Y. Ogras, "Predictive dynamic thermal and power management for heterogeneous mobile platforms," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*. IEEE, 2015, pp. 960–965.

[28] M. Fan, R. Rong, S. Liu, and G. Quan, "Energy calculation for periodic multi-core scheduling in system thermal steady state with consideration of leakage and temperature dependency," *The Journal of Supercomputing*, vol. 71, no. 7, pp. 2565–2584, 2015.

[29] 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*. IEEE, 2012, pp. 408–415.

[30] Q. Han, T. Wang, and G. Quan, "Enhanced fault-tolerant fixed-priority scheduling of hard real-time tasks on multi-core platforms," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2015 IEEE 21st International Conference on*. IEEE, 2015, pp. 21–30.

[31] R. Nair, "Evolution of memory architecture," *Proceedings of the IEEE*, vol. 103, no. 8, pp. 1331–1345, 2015.

[32] M. Greenberg, "DRAM in the Automobilie: what, where, why, and how," 2016. [Online]. Available: http://www.memcon.com/pdfs/proceedings2016/KEY102.pdf

[33] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, 2016.

[34] W. A. Wulf and S. A. McKee, "Hitting the memory wall: implications of the obvious," *ACM SIGARCH computer architecture news*, vol. 23, no. 1, pp. 20–24, 1995.

[35] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2012.

[36] G. Yao, R. Pellizzoni, S. Bak, E. Betti, and M. Caccamo, "Memory-centric scheduling for multicore hard real-time systems," *Real-Time Systems*, vol. 48, no. 6, pp. 681–715, 2012.

[37] S. A. McKee, "Reflections on the memory wall," in *Proceedings of the 1st conference on Computing frontiers*. ACM, 2004, p. 162.

[38] O. Mutlu and L. Subramanian, "Research problems and opportunities in memory systems," *Supercomputing frontiers and innovations*, vol. 1, no. 3, pp. 19–55, 2015.

[39] L. A. D. Bathen and N. D. Dutt, "Software controlled memories for scalable many-core architectures," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2012 IEEE 18th International Conference on*. IEEE, 2012, pp. 1–10.

[40] O. Kotaba, J. Nowotsch, M. Paulitsch, S. M. Petters, and H. Theiling, "Multi-core in real-time systems–temporal isolation challenges due to shared resources," in *Workshop on Industry-Driven Approaches for Cost-effective Certification of Safety-Critical, Mixed-Criticality Systems*, 2013.

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

[42] J. Lin, H. Zheng, Z. Zhu, and Z. Zhang, "Thermal modeling and management of DRAM systems," *IEEE Transactions on Computers*, vol. 62, no. 10, pp. 2069–2082, 2013.

[43] M. Benson, *The art of software thermal management for embedded systems*. Springer, 2014.

[44] M. Bambagini, M. Marinoni, H. Aydin, and G. Buttazzo, "Energy-aware scheduling for real-time systems: a survey," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 15, no. 1, p. 7, 2016.

[45] L. Cheng, K. Huang, G. Chen, B. Hu, and A. Knoll, "Periodic thermal management for hard real-time systems," in *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*. IEEE, 2015, pp. 1–10.

[46] M. Ahmed, N. Fisher, S. Wang, and P. Hettiarachchi, "Minimizing peak temperature in embedded real-time systems via thermal-aware periodic resources," *Sustainable Computing: Informatics and Systems (SUSCOM)*, vol. 1, no. 3, pp. 226–240, 2011.

[47] L. Minas and B. Ellison, "The problem of power consumption in servers," *Intel Corporation*, 2009.

[48] L. A. Barroso, J. Clidaras, and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis lectures on computer architecture*, vol. 8, no. 3, pp. 1–154, 2013.

[49] S. Mittal, "A survey of techniques for improving energy efficiency in embedded computing systems," *International Journal of Computer Aided Engineering and Technology*, vol. 6, no. 4, pp. 440–459, 2014.

[50] A. Suresh, P. Cicotti, and L. Carrington, "Evaluation of emerging memory technologies for hpc, data intensive applications," in *Cluster Computing (CLUSTER), 2014 IEEE International Conference on*. IEEE, 2014, pp. 239–247.

[51] O. Mutlu, J. Meza, and L. Subramanian, "The main memory system: Challenges and opportunities," *Communications of the Korean Institute of Information Scientists and Engineers*, 2015.

[52] J. Boukhobza, S. Rubini, R. Chen, and Z. Shao, "Emerging NVM: A Survey on Architectural Integration and Research Challenges," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 23, no. 2, p. 14, 2017.

[53] S. M. Hassan, "Exploiting On-Chip Memory Concurrency in 3D Manycore Architectures," Ph.D. dissertation, Georgia Institute of Technology, 2016.

[54] J. Meng and A. K. Coskun, "Analysis and runtime management of 3D systems with stacked DRAM for boosting energy efficiency," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012*. IEEE, 2012, pp. 611–616.

[55] A. Rahmati, M. Hicks, D. Holcomb, and K. Fu, "Refreshing thoughts on DRAM: Power saving vs. data integrity," in *Workshop on Approximate Computing Across the System Stack (WACAS)*, 2014.

[56] D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu, "Adaptive-latency DRAM: Optimizing DRAM timing for the common-case," in

High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on. IEEE, 2015, pp. 489–501.

[57] H. Zhang, X. Zhang, B. Lau, S. Lim, L. Ding, M. Yu, and Y. Lee, "Thermal characterization and simulation study of 2.5D packages with multi-chip module on through silicon interposer," in *Electronics Packaging Technology Conference (EPTC 2013), 2013 IEEE 15th*. IEEE, 2013, pp. 363–368.

[58] K. Chandrasekar, C. Weis, B. Akesson, N. Wehn, and K. Goossens, "System and circuit level power modeling of energy-efficient 3D-stacked wide I/O DRAMs," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013*. IEEE, 2013, pp. 236–241.

[59] Y.-J. Chen, C.-L. Yang, P.-S. Lin, and Y.-C. Lu, "Thermal/performance characterization of CMPs with 3D-stacked DRAMs under synergistic voltage-frequency control of cores and DRAMs," in *Proceedings of the 2015 Conference on research in adaptive and convergent systems*. ACM, 2015, pp. 430–436.

[60] R. Ayoub, K. R. Indukuri, and T. S. Rosing, "Energy efficient proactive thermal management in memory subsystem," in *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on*. IEEE, 2010, pp. 195–200.

[61] S. Liu, B. Leung, A. Neckar, S. O. Memik, G. Memik, and N. Hardavellas, "Hardware/software techniques for DRAM thermal management," in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*. IEEE, 2011, pp. 515–525.

[62] G. Gracioli and A. A. Frohlich, "An experimental evaluation of the cache partitioning impact on multicore real-time schedulers," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2013 IEEE 19th International Conference on*. IEEE, 2013, pp. 72–81.

[63] H. Hanson and K. Rajamani, "What computer architects need to know about memory throttling," in *International Symposium on Computer Architecture*. Springer, 2010, pp. 233–242.

[64] S. Siewert, *Real-time embedded components and systems*. Cengage Learning, 2006.

[65] S. Baruah and K. Pruhs, "Open problems in real-time scheduling," *Journal of Scheduling*, vol. 13, no. 6, pp. 577–582, 2010.

[66] F. Rammig, M. Ditze, P. Janacik, T. Heimfarth, T. Kerstan, S. Oberthuer, and K. Stahl, "Basic concepts of real time operating systems," in *Hardware-dependent Software*. Springer, 2009, pp. 15–45.

[67] J. A. Stankovic, "Misconceptions about real-time computing: A serious problem for next-generation systems," *Computer*, vol. 21, no. 10, pp. 10–19, 1988.

[68] A. Burns and R. Davis, "Mixed criticality systems-a review," *Department of Computer Science, University of York, Tech. Rep*, pp. 1–69, 2013.

[69] G. Gracioli, A. A. Fröhlich, R. Pellizzoni, and S. Fischmeister, "Implementation and evaluation of global and partitioned scheduling in a real-time OS," *Real-Time Systems*, vol. 49, no. 6, pp. 669–714, 2013.

[70] Y. Oh and S. H. Son, "Allocating fixed-priority periodic tasks on multiprocessor systems," *Real-Time Systems*, vol. 9, no. 3, pp. 207–239, 1995.

[71] G. C. Buttazzo, "Rate monotonic vs. EDF: judgment day," *Real-Time Systems*, vol. 29, no. 1, pp. 5–26, 2005.

[72] J. M. Rivas, J. J. Gutiérrez, and M. G. Harbour, "Fixed priorities or EDF for distributed real-time systems?" *ACM SIGBED Review*, vol. 10, no. 2, pp. 21–21, 2013.

[73] J. C. Palencia and M. G. Harbour, "Response time analysis of EDF distributed real-time systems," *Journal of Embedded Computing*, vol. 1, no. 2, pp. 225–237, 2005.

[74] A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son, "New strategies for assigning real-time tasks to multiprocessor systems," *IEEE transactions on computers*, vol. 44, no. 12, pp. 1429–1442, 1995.

[75] C.-C. Han and H.-Y. Tyan, "A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithms," in *Real-Time Systems Symposium, 1997. Proceedings., The 18th IEEE.* IEEE, 1997, pp. 36–45.

[76] 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.* IEEE, 2014, pp. 284–291.

[77] L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok, "Real time scheduling theory: A historical perspective," *Real-time systems*, vol. 28, no. 2-3, pp. 101–155, 2004.

[78] T. P. Baker, "A comparison of global and partitioned EDF schedulability tests for multiprocessors," in *In International Conf. on Real-Time and Network Systems*. Citeseer, 2005.

[79] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*. IEEE, 2003, pp. 2–13.

[80] C. Guo, X. Hua, H. Wu, D. Lautner, and S. Ren, "Best-harmonically-fit periodic task assignment algorithm on multiple periodic resources," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1303–1315, 2016.

[81] J. Hsu, "Power problems threaten to strangle exascale computing," 2015. [Online]. Available: https://spectrum.ieee.org/computing/hardware/power-problems-threaten-to-strangle-exascale-computing

[82] K. Zhang, A. Guliani, S. Ogrenci-Memik, G. Memik, K. Yoshii, R. Sankaran, and P. Beckman, "Machine learning-based temperature prediction for runtime thermal management across system components," *IEEE Transactions on Parallel and Distributed Systems*, 2017.

[83] Q. Zhang and W. Shi, "Energy-efficient workload placement in enterprise datacenters," *Computer*, vol. 49, no. 2, pp. 46–52, 2016.

[84] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 732–794, 2016.

[85] M. T. Chaudhry, T. C. Ling, A. Manzoor, S. A. Hussain, and J. Kim, "Thermal-aware scheduling in green data centers," *ACM Computing Surveys (CSUR)*, vol. 47, no. 3, p. 39, 2015.

[86] T. Patki, D. K. Lowenthal, A. Sasidharan, M. Maiterth, B. L. Rountree, M. Schulz, and B. R. De Supinski, "Practical resource management in power-constrained, high performance computing," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2015, pp. 121–132.

[87] M. Maiterth, T. Wilde, D. Lowenthal, B. Rountree, M. Schulz, J. Eastep, and D. Kranzlmiiller, "Power aware high performance computing: Challenges and opportunities for application and system developers - survey & tutorial," in *High Performance Computing & Simulation (HPCS), 2017 International Conference on*. IEEE, 2017, pp. 3–10.

[88] V. Chaturvedi, H. Huang, and G. Quan, "Leakage aware scheduling on maximum temperature minimization for periodic hard real-time systems," in *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*. IEEE, 2010, pp. 1802–1809.

[89] I. Ukhov, M. Bao, P. Eles, and Z. Peng, "Steady-state dynamic temperature analysis and reliability optimization for embedded multiprocessor systems," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 197–204.

[90] G. Quan and V. Chaturvedi, "Feasibility analysis for temperature-constraint hard real-time periodic tasks," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 3, pp. 329–339, 2010.

[91] K. Skadron, T. Abdelzaher, and M. R. Stan, "Control-theoretic techniques and thermal-rc modeling for accurate and localized dynamic thermal management," in *High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium on*. IEEE, 2002, pp. 17–28.

[92] B. Zhao, H. Aydin, and D. Zhu, "Generalized reliability-oriented energy management for real-time embedded applications," in *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*. IEEE, 2011, pp. 381–386.

[93] A. A. Fröhlich, "A comprehensive approach to power management in embedded systems," *International Journal of Distributed Sensor Networks*, vol. 7, no. 1, p. 807091, 2011.

[94] G. A. Chaparro-Baquero, Q. Zhou, C. Liu, J. Tang, and S. Liu, "Power-Efficient Schemes Via Workload Characterization on the Intel's Single-chip Cloud Computer," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*. IEEE, 2012, pp. 999–1006.

[95] B. Zhao, H. Aydin, and D. Zhu, "Energy management under general task-level reliability constraints," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2012 IEEE 18th*. IEEE, 2012, pp. 285–294.

[96] M. A. Awan, P. M. Yomsi, G. Nelissen, and S. M. Petters, "Energy-aware task mapping onto heterogeneous platforms using DVFS and sleep states," *Real-Time Systems*, vol. 52, no. 4, pp. 450–485, 2016.

[97] X. Pan, W. Jiang, K. Jiang, L. Wen, and Q. Dong, "Energy optimization of stochastic applications with statistical guarantees of deadline and reliability," in *Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific*. IEEE, 2016, pp. 324–329.

[98] M. K. Bhatti, C. Belleudy, and M. Auguin, "Hybrid power management in real time embedded systems: an interplay of dvfs and dpm techniques," *Real-Time Systems*, vol. 47, no. 2, pp. 143–162, 2011.

[99] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo, "Applying real-time interface and calculus for dynamic power management in hard real-time systems," *Real-Time Systems*, vol. 47, no. 2, pp. 163–193, 2011.

[100] G. Terzopoulos and H. Karatza, "Performance evaluation and energy consumption of a real-time heterogeneous grid system using dvs and dpm," *Simulation Modelling Practice and Theory*, vol. 36, pp. 33–43, 2013.

[101] Q. Han, M. Fan, L. Niu, and G. Quan, "Energy minimization for fault tolerant scheduling of periodic fixed-priority applications on multiprocessor platforms," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 830–835.

[102] Q. Han, M. Fan, O. Bai, S. Ren, and G. Quan, "Temperature-constrained feasibility analysis for multicore scheduling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 12, pp. 2082–2092, 2016.

[103] T. Chantem, X. S. Hu, and R. P. Dick, "Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 10, pp. 1884–1897, 2011.

[104] O. Sahin, P. T. Varghese, and A. K. Coskun, "Just enough is more: Achieving sustainable performance in mobile devices under thermal limitations," in *Computer-Aided Design (ICCAD), 2015 IEEE/ACM International Conference on*. IEEE, 2015, pp. 839–846.

[105] J.-J. Chen, S. Wang, and L. Thiele, "Proactive speed scheduling for real-time tasks under thermal constraints," in *Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE*. IEEE, 2009, pp. 141–150.

[106] H. Huang, V. Chaturvedi, G. Quan, J. Fan, and M. Qiu, "Throughput maximization for periodic real-time systems under the maximal temperature constraint," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 2s, p. 70, 2014.

[107] B. Egilmez, G. Memik, S. Ogrenci-Memik, and O. Ergin, "User-specific skin temperature-aware DVFS for smartphones," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 1217–1220.

[108] S. Sha, W. Wen, M. Fan, S. Ren, and G. Quan, "Performance maximization via frequency oscillation on temperature constrained multi-core processors," in *Parallel Processing (ICPP), 2016 45th International Conference on*. IEEE, 2016, pp. 526–535.

[109] P. M. Hettiarachchi, N. Fisher, M. Ahmed, L. Y. Wang, S. Wang, and W. Shi, "A design and analysis framework for thermal-resilient hard real-time systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 5s, p. 146, 2014.

[110] J. V. Wang, C.-T. Cheng, and K. T. Chi, "A power and thermal-aware virtual machine allocation mechanism for cloud data centers," in *Communication Workshop (ICCW), 2015 IEEE International Conference on*. IEEE, 2015, pp. 2850–2855.

[111] K. Zhang, S. Ogrenci-Memik, G. Memik, K. Yoshii, R. Sankaran, and P. Beckman, "Minimizing thermal variation across system components," in *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*. IEEE, 2015, pp. 1139–1148.

[112] B. D. Bui, M. Caccamo, L. Sha, and J. Martinez, "Impact of cache partitioning on multi-tasking real time embedded systems," in *Embedded and Real-Time Computing Systems and Applications, 2008. RTCSA'08. 14th IEEE International Conference on*. IEEE, 2008, pp. 101–110.

[113] S. P. Muralidhara, M. Kandemir, and P. Raghavan, "Intra-application cache partitioning," in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–12.

[114] J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, and P. Sadayappan, "Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems," in *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*. IEEE, 2008, pp. 367–378.

[115] Y. Chen, W. Li, C. Kim, and Z. Tang, "Efficient shared cache management through sharing-aware replacement and streaming-aware insertion policy," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on.* IEEE, 2009, pp. 1–11.

[116] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on.* IEEE, 2006, pp. 423–432.

[117] J. Liedtke, H. Hartig, and M. Hohmuth, "OS-controlled cache predictability for real-time systems," in *Real-Time Technology and Applications Symposium, 1997. Proceedings., Third IEEE.* IEEE, 1997, pp. 213–224.

[118] R. Mancuso, R. Dudko, E. Betti, M. Cesati, M. Caccamo, and R. Pellizzoni, "Real-time cache management framework for multi-core architectures," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th.* IEEE, 2013, pp. 45–54.

[119] N. Guan, M. Stigge, W. Yi, and G. Yu, "Cache-aware scheduling and analysis for multicores," in *Proceedings of the seventh ACM international conference on Embedded software.* ACM, 2009, pp. 245–254.

[120] A. Chousein and R. N. Mahapatra, "Fully associative cache partitioning with don't care bits for real-time applications," *ACM SIGBED Review*, vol. 2, no. 2, pp. 35–38, 2005.

[121] M. Shekhar, A. Sarkar, H. Ramaprasad, and F. Mueller, "Semi-partitioned hard-real-time scheduling under locked cache migration in multicore systems," in *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on.* IEEE, 2012, pp. 331–340.

[122] X. Vera, B. Lisper, and J. Xue, "Data caches in multitasking hard real-time systems," in *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE.* IEEE, 2003, pp. 154–165.

[123] D. Sanchez and C. Kozyrakis, "Vantage: scalable and efficient fine-grain cache partitioning," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 3, pp. 57–68, 2011.

[124] A. Sarkar, F. Mueller, and H. Ramaprasad, "Static task partitioning for locked caches in multicore real-time systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 14, no. 1, p. 4, 2015.

[125] B. C. Ward, J. L. Herman, C. J. Kenna, and J. H. Anderson, "Outstanding paper award: Making shared caches more predictable on multicore platforms," in *Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on*. IEEE, 2013, pp. 157–167.

[126] B. Lesage, I. Puaut, and A. Seznec, "PRETI: Partitioned REal-TIme shared cache for mixed-criticality real-time systems," in *Proceedings of the 20th International Conference on Real-Time and Network Systems*. ACM, 2012, pp. 171–180.

[127] N. Suzuki, H. Kim, D. De Niz, B. Andersson, L. Wrage, M. Klein, and R. Rajkumar, "Coordinated bank and cache coloring for temporal protection of memory accesses," in *Computational Science and Engineering (CSE), 2013 IEEE 16th International Conference on*. IEEE, 2013, pp. 685–692.

[128] J. Meng, K. Kawakami, and A. K. Coskun, "Optimizing energy efficiency of 3D multicore systems with stacked DRAM under power and thermal constraints," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 648–655.

[129] J. Ji, C. Wang, and X. Zhou, "System-level early power estimation for memory subsystem in embedded systems," in *Embedded Computing, 2008. SEC'08. Fifth IEEE International Symposium on*. IEEE, 2008, pp. 370–375.

[130] MICRON, "Calculating Memory System Power for DDR3 - Technical Note," Micron Technol. Inc., Tech. Rep., 2007.

[131] H. Yun, R. Mancuso, Z.-P. Wu, and R. Pellizzoni, "PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2014 IEEE 20th*. IEEE, 2014, pp. 155–166.

[132] L. Liu, Z. Cui, M. Xing, Y. Bao, M. Chen, and C. Wu, "A software memory partition approach for eliminating bank-level interference in multicore systems," in *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*. ACM, 2012, pp. 367–376.

[133] W. Mi, X. Feng, J. Xue, and Y. Jia, "Software-hardware cooperative dram bank partitioning for chip multiprocessors," in *IFIP International Conference on Network and Parallel Computing*. Springer, 2010, pp. 329–343.

[134] J. Reineke, I. Liu, H. D. Patel, S. Kim, and E. A. Lee, "PRET DRAM controller: Bank privatization for predictability and temporal isolation," in *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2011 Proceedings of the 9th International Conference on*. IEEE, 2011, pp. 99–108.

[135] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th*. IEEE, 2013, pp. 55–64.

[136] R. Inam, J. Slatman, M. Behnam, M. Sjödin, and T. Nolte, "Towards implementing multi-resource server on multi-core linux platform," in *Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on*. IEEE, 2013, pp. 1–4.

[137] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via dynamic voltage/frequency scaling," in *Proceedings of the 8th ACM international conference on Autonomic computing*. ACM, 2011, pp. 31–40.

[138] Q. Deng, D. Meisner, A. Bhattacharjee, T. F. Wenisch, and R. Bianchini, "MultiScale: memory system DVFS with multiple memory controllers," in *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*. ACM, 2012, pp. 297–302.

[139] R. Begum, D. Werner, M. Hempstead, G. Prasad, and G. Challen, "Energy-performance trade-offs on energy-constrained devices with multi-component DVFS," in *Workload Characterization (IISWC), 2015 IEEE International Symposium on*. IEEE, 2015, pp. 34–43.

[140] I. Anagnostopoulos, J.-M. Chabloz, I. Koutras, A. Bartzas, A. Hemani, and D. Soudris, "Power-aware dynamic memory management on many-core platforms utilizing DVFS," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 1s, p. 40, 2013.

[141] R. Miftakhutdinov, E. Ebrahimi, and Y. N. Patt, "Predicting performance impact of DVFS for realistic memory systems," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2012, pp. 155–165.

[142] W. dos Santos Marques, P. S. S. de Souza, A. F. Lorenzon, A. C. S. Beck, M. B. Rutzig, and F. D. Rossi, "Improving EDP in multi-core embedded systems through multidimensional frequency scaling," in *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 1–4.

[143] G. Jia, X. Li, J. Wan, C. Wang, D. Dai, and C. Jiang, "Coordinate task and memory management for improving power efficiency," in *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 2013, pp. 267–278.

[144] I. Paul, W. Huang, M. Arora, and S. Yalamanchili, "Harmonia: balancing compute and memory power in high-performance GPUs," in *Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on*. IEEE, 2015, pp. 54–65.

[145] A. Tiwari, M. Schulz, and L. Carrington, "Predicting optimal power allocation for CPU and DRAM domains," in *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*. IEEE, 2015, pp. 951–959.

[146] R. Begum, M. Hempstead, G. P. Srinivasa, and G. Challen, "Algorithms for CPU and DRAM DVFS under inefficiency constraints," in *Computer Design (ICCD), 2016 IEEE 34th International Conference on*. IEEE, 2016, pp. 161–168.

[147] J. Jang and M. Park, "DRAM frequency scaling for energy efficiency based on memory usage," in *Consumer Electronics (ICCE), 2017 IEEE International Conference on*. IEEE, 2017, pp. 308–309.

[148] A. Agrawal, G. Fohler, J. Freitag, J. Nowotsch, S. Uhrig, and M. Paulitsch, "Contention-Aware Dynamic Memory Bandwidth Isolation With Predictability in COTS Multicores: An Avionics Case Study," in *LIPIcs-Leibniz International Proceedings in Informatics*, vol. 76. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[149] L. Siddhu and P. R. Panda, "Thermal aware runtime management of 3D memory architecture," *CSI transactions on ICT*, vol. 5, no. 2, pp. 129–134, 2017.

[150] N. Kumari, R. Shih, S. Escobar-Vargas, T. Cader, A. Govyadinov, S. Anthony, and C. Bash, "Air cooling limits of 3D stacked logic processor and memory dies," in *Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), 2014 IEEE Intersociety Conference on*. IEEE, 2014, pp. 92–97.

[151] G. L. Loi, B. Agrawal, N. Srivastava, S.-C. Lin, T. Sherwood, and K. Banerjee, "A thermally-aware performance analysis of vertically integrated (3-D) processor-

memory hierarchy," in *Design Automation Conference, 2006 43rd ACM/IEEE*. IEEE, 2006, pp. 991–996.

[152] G. H. Loh, "3D-stacked memory architectures for multi-core processors," in *Computer Architecture, 2008. ISCA'08. 35th International Symposium on*. IEEE, 2008, pp. 453–464.

[153] Y. Zhang, L. Li, Z. Lu, A. Jantsch, M. Gao, H. Pan, and F. Han, "A survey of memory architecture for 3d chip multi-processors," *Microprocessors and Microsystems*, vol. 38, no. 5, pp. 415–430, 2014.

[154] Y.-J. Chen, C.-L. Yang, P.-S. Lin, and Y.-C. Lu, "Opportunities of synergistically adjusting voltage-frequency levels of cores and DRAMs in CMPs with 3d-stacked DRAMs for efficient thermal control," *ACM SIGAPP Applied Computing Review*, vol. 16, no. 1, pp. 26–35, 2016.

[155] B. K. Mohanty, V. Chaturvedi, V. Rathore, and T. Srikanthan, "Memory-access aware work-load distribution for peak-temperature reduction of 3D multi-core embedded systems," in *Digital Signal Processing (DSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1270–1273.

[156] S.-Y. Lin and J.-Y. Lin, "Thermal-aware architecture and mapping for multi-channel three-dimensional DRAM systems," in *Consumer Electronics (GCCE), 2014 IEEE 3rd Global Conference on*. IEEE, 2014, pp. 713–714.

[157] D. Zhao, H. Homayoun, and A. V. Veidenbaum, "Temperature aware thread migration in 3D architecture with stacked DRAM," in *Quality Electronic Design (ISQED), 2013 14th International Symposium on*. IEEE, 2013, pp. 80–87.

[158] C.-W. Chang, J.-J. Chen, T.-W. Kuo, and H. Falk, "Real-time partitioned scheduling on multi-core systems with local and global memories," in *Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific*. IEEE, 2013, pp. 467–472.

[159] F. Eisenbrand, N. Hähnle, M. Niemeier, M. Skutella, J. Verschae, and A. Wiese, "Scheduling periodic tasks in a hard real-time environment," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2010, pp. 299–311.

[160] M. B. Taylor, "A landscape of the new dark silicon design regime," *IEEE Micro*, vol. 33, no. 5, pp. 8–19, 2013.

[161] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*. IEEE, 2011, pp. 365–376.

[162] J. Rosén, P. Eles, Z. Peng, and A. Andrei, "Predictable worst-case execution time analysis for multiprocessor systems-on-chip," in *Electronic Design, Test and Application (DELTA), 2011 Sixth IEEE International Symposium on*. IEEE, 2011, pp. 99–104.

[163] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra *et al.*, "The worst-case execution-time problem overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 3, p. 36, 2008.

[164] L. R. Hsu, S. K. Reinhardt, R. Iyer, and S. Makineni, "Communist, utilitarian, and capitalist cache policies on CMPs: caches as a shared resource," in *Parallel Architectures and Compilation Techniques (PACT), 2006 International Conference on*. IEEE, 2006, pp. 13–22.

[165] R. Iyer, L. Zhao, F. Guo, R. Illikkal, S. Makineni, D. Newell, Y. Solihin, L. Hsu, and S. Reinhardt, "QoS policies and architecture for cache/memory in CMP platforms," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 35, no. 1. ACM, 2007, pp. 25–36.

[166] D. Tam, R. Azimi, L. Soares, and M. Stumm, "Managing shared L2 caches on multicore systems in software," in *Workshop on the Interaction between Operating Systems and Computer Architecture*. Citeseer, 2007, pp. 26–33.

[167] V. Suhendra and T. Mitra, "Exploring locking & partitioning for predictable shared caches on multi-cores," in *Proceedings of the 45th annual Design Automation Conference*. ACM, 2008, pp. 300–303.

[168] S. Altmeyer, R. Douma, W. Lunniss, and R. Davis, "Evaluation of cache partitioning for hard real-time systems," in *proceedings Euromicro Conference on Real-Time Systems (ECRTS)*, 2014, pp. 15–26.

[169] B. Berna and I. Puaut, "PDPA: period driven task and cache partitioning algorithm for multi-core systems," in *Proceedings of the 20th International Conference on Real-Time and Network Systems*. ACM, 2012, pp. 181–189.

[170] M. Paolieri, E. Quiñones, F. J. Cazorla, R. I. Davis, and M. Valero, "IA^3: An interference aware allocation algorithm for multicore hard real-time systems," in

Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE. IEEE, 2011, pp. 280–290.

[171] H. Kim, A. Kandhalu, and R. Rajkumar, "A coordinated approach for practical OS-level cache management in multi-core real-time systems," in *Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on*. IEEE, 2013, pp. 80–89.

[172] M. Fan and G. Quan, "Harmonic-aware multi-core scheduling for fixed-priority real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1476–1488, 2014.

[173] J. F. Cantin and M. D. Hill, "Cache performance for selected SPEC CPU2000 benchmarks," *ACM SIGARCH Computer Architecture News*, vol. 29, no. 4, pp. 13–18, 2001.

[174] J. L. Henning, "SPEC CPU2000: Measuring CPU performance in the new millennium," *Computer*, vol. 33, no. 7, pp. 28–35, 2000.

[175] D. Burger and T. M. Austin, "The simplescalar tool set, version 2.0," *ACM SIGARCH computer architecture news*, vol. 25, no. 3, pp. 13–25, 1997.

[176] Y. Zhang, L. Peng, B. Li, J.-K. Peir, and J. Chen, "Architecture comparisons between Nvidia and ATI GPUs: Computation parallelism and data communications," in *Workload Characterization (IISWC), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 205–215.

[177] Y. Iwase, D. Abe, and T. Yakoh, "GPGPU aided method for real-time systems," in *Industrial Informatics (INDIN), 2012 10th IEEE International Conference on*. IEEE, 2012, pp. 841–845.

[178] R. L. Mason, R. F. Gunst, and J. L. Hess, *Statistical design and analysis of experiments: with applications to engineering and science*. John Wiley & Sons, 2003, vol. 474.

[179] L. Wilkinson, "Revising the Pareto chart," *The American Statistician*, vol. 60, no. 4, pp. 332–334, 2006.

[180] R. Ayoub, R. Nath, and T. S. Rosing, "CoMETC: Coordinated management of energy/thermal/cooling in servers," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 19, no. 1, p. 1, 2013.

[181] C. Zhu, Z. Gu, L. Shang, R. P. Dick, and R. Joseph, "Three-dimensional chip-multiprocessor run-time thermal management," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 8, pp. 1479–1492, 2008.

[182] S. Pagani, J.-J. Chen, M. Shafique, and J. Henkel, "MatEx: efficient transient and peak temperature computation for compact thermal models," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 2015, pp. 1515–1520.

[183] S. Sha, W. Wen, S. Ren, and G. Quan, "M-oscillating: Performance maximization on temperature-constrained multi-core processors," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2018, (Under review).

[184] MICRON, "MICRON 2Gb DDR3 SDRAM," Micron Technol. Inc., Tech. Rep., 2006.

[185] L. Yuan, S. Leventhal, and G. Qu, "Temperature-aware leakage minimization technique for real-time systems," in *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*. ACM, 2006, pp. 761–764.

[186] M. Jung, É. Zulian, D. M. Mathew, M. Herrmann, C. Brugger, C. Weis, and N. Wehn, "Omitting Refresh: A Case Study for Commodity and Wide I/O DRAMs," in *Proceedings of the 2015 International Symposium on Memory Systems*. ACM, 2015, pp. 85–91.

[187] W.-K. Cheng and T.-W. Hsu, "Thermal-aware task allocation, memory mapping, and task scheduling for 3d stacked memory and processor architecture," in *TENCON Spring Conference, 2013 IEEE*. IEEE, 2013, pp. 95–98.

[188] V. Sundriyal and M. Sosonkina, "Joint frequency scaling of processor and DRAM," *The Journal of Supercomputing*, vol. 72, no. 4, pp. 1549–1569, 2016.

[189] T.-H. Tsai and Y.-S. Chen, "Thermal-throttling server: A thermal-aware real-time task scheduling framework for three-dimensional multicore chips," *Journal of Systems and Software*, vol. 112, pp. 11–25, 2016.

[190] X. Zhou, J. Yang, Y. Xu, Y. Zhang, and J. Zhao, "Thermal-aware task scheduling for 3D multicore processors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 1, pp. 60–71, 2010.

[191] R. Ge, X. Feng, Y. He, and P. Zou, "The case for cross-component power coordination on power bounded systems," in *Parallel Processing (ICPP), 2016 45th International Conference on*. IEEE, 2016, pp. 516–525.

[192] J. Meng, D. Rossell, and A. K. Coskun, "Exploring performance, power, and temperature characteristics of 3D systems with on-chip DRAM," in *Green Computing Conference and Workshops (IGCC), 2011 International*. IEEE, 2011, pp. 1–6.

[193] J. Rosen, A. Andrei, P. Eles, and Z. Peng, "Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip," in *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*. IEEE, 2007, pp. 49–60.

[194] G. A. Chaparro-Baquero, S. Sha, S. Homsi, W. Wen, and G. Quan, "Processor/memory co-scheduling using periodic resource server for real-time systems under peak temperature constraints," in *Quality Electronic Design (ISQED), 2017 18th International Symposium on*. IEEE, 2017, pp. 360–366.

[195] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.

[196] R. Inam and M. Sjödin, "Combating unpredictability in multicores through the multi-resource server," in *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*. IEEE, 2014, pp. 1–8.

[197] R. Ishihara, J. Derakhshandeh, M. T. Mofrad, T. Chen, N. Golshani, and C. Beenakker, "Monolithic 3d-ics with single grain si thin film transistors," *Solid-State Electronics*, vol. 71, pp. 80–87, 2012.

[198] MONOLITHIC3D.COM, "Qualcomm to leverage Monolithic 3D to win Smartphone Market Share," April 2015. [Online]. Available: http://www.monolithic3d.com/blog/qualcomm-to-leverage-monolithic-3d-to-win-smartphone-market-share

[199] S. Panth, K. Samadi, Y. Du, and S. K. Lim, "Shrunk-2D: A Physical Design Methodology to Build Commercial-Quality Monolithic 3D ICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[200] S. K. Samal, S. Panth, K. Samadi, M. Saedi, Y. Du, and S. K. Lim, "Fast and accurate thermal modeling and optimization for monolithic 3D ICs," in *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*. IEEE, 2014, pp. 1–6.

[201] M. Saeidi, K. Samadi, A. Mittal, and R. Mittal, "Thermal implications of mobile 3D-ICs," in *3D Systems Integration Conference (3DIC), 2014 International.* IEEE, 2014, pp. 1–7.

[202] Y. Yu and N. K. Jha, "Energy-Efficient Monolithic 3D on-Chip Memory Architectures," *IEEE Transactions on Nanotechnology*, 2017.

[203] Y. Turakhia, G. Liu, S. Garg, and D. Marculescu, "Thread progress equalization: Dynamically adaptive power-constrained performance optimization of multi-threaded applications," *IEEE Transactions on Computers*, vol. 66, no. 4, pp. 731–744, 2017.

[204] G. H. Loh, N. E. Jerger, A. Kannan, and Y. Eckert, "Interconnect-memory challenges for multi-chip, silicon interposer systems," in *Proceedings of the 2015 International Symposium on Memory Systems.* ACM, 2015, pp. 3–10.

[205] A. Sodani, "Knights landing (KNL): 2nd Generation Intel® Xeon Phi processor," in *Hot Chips 27 Symposium (HCS), 2015 IEEE.* IEEE, 2015, pp. 1–24.

VITA

GUSTAVO A. CHAPARRO-BAQUERO

2002	B.S., Electronics Engineering Pontifical Xavierian University Bogotá, Colombia
2007	M.S., Computer Engineering University of Puerto Rico at Mayagüez (UPRM) Mayagüez, Puerto Rico
2018	Ph.D., Electrical and Computer Engineering Florida International University (FIU) Miami, Florida

PUBLICATIONS

- Chaparro-Baquero G., Sha S., Homsi S., Wen W., Quan G. "Thermal-aware Joint CPU and Memory Scheduling for Hard Real-Time Tasks on Multicore 3D Platforms." On 8th IEEE International Green and Sustainable Computing Conference (IGSC). 2017.
- Chaparro-Baquero G., Sha S., Homsi S., Wen W., Quan G. "Processor/Memory Co-scheduling Using Periodic Resource Server for Real-Time System Under Peak Temperature Constraints." On 18th IEEE International Symposium on Quality Electronic Design (ISQED). 2017.
- Homsi S., Liu S., Chaparro-Baquero G., Bai O., Ren S., Quan G. "Workload Consolidation for Cloud Data Centers with Guaranteed QoS Using Request Reneging." On IEEE Transactions on Parallel and Distributed Systems.
doi: 10.1109/TPDS.2016.2642941. 2016.
- Chaparro-Baquero G., Homsi S., Vichot O., Ren S. Quan G., Ren S. "Cache Allocation for Fixed-Priority Real-Time Scheduling on Multi-Core Platforms." On 33rd IEEE International Conference on Computer Design (ICCD). 2015.
- Chaparro-Baquero, G., Zhou Q., Liu C., Tang J., and Liu S. "Power-Efficient Schemes via Workload Characterization on the Intel's Single-Chip Cloud Computer." On 26th IEEE International Parallel and Distributed Processing Symposium Workshop & Ph.D. Forum (IPDPSW). 2012.
- Chaparro-Baquero, Gustavo A. "Petri-Net Workflow Modeling for Digital Publishing - Measuring Quantitative Dependability Attributes." VDM Verlag Dr. Müller. 2007. ISBN-10: 3836418894. ISBN-13: 978-3836418898.