UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

# Energy Efficient Scheduling for Hard Real-Time Systems

---

## Dissertation

A thesis submitted to the
Faculty of Computer Science,
Electrical Engineering and Mathematics
of the
University of Paderborn
in partial fulfillment
of the requirements for the degree of Dr. rer. nat.

by

**Da He**

Paderborn

*Supervisors:*
**Prof. Dr. Franz Josef Rammig (Universität Paderborn)**
**Prof. Dr. Achim Rettberg (Carl von Ossietzky Universität Oldenburg)**

**Date of public examination: December 20, 2013.**

# Abstract

In modern electronic systems, especially in battery-driven devices, energy consumption has clearly become one of the most important design concerns. Low power consumption and long battery life are major development requirements and objectives to reduce system operation cost. From the system-level point of view, there are two widely applied energy saving techniques, Dynamic Power Management (DPM) and Dynamic Voltage and Frequency Scaling (DVS), which are able to adjust the trade-off between system performance and power consumption. In brief, DPM tries to selectively shut down unused components, whereas DVS attempts to slow down them. In fact, both techniques reduce system power consumption at the cost of performance loss, which is a crucial point in hard real-time systems.

To address energy optimization problem, this dissertation studies in detail the combined application of DPM and DVS on both single- and multi-core processor platforms, in particular with non-negligible state switching overhead. Unfortunately, the facing problem is proven to be $\mathcal{NP}$-hard in the strong sense, which indicates non-existence of efficient algorithms. Thus, this work proposes a heuristic search algorithm by extending simulated annealing with neighbor selection guidelines using domain specific information. In addition, a regression based mechanism to predict algorithm run-time behavior is proposed, which in turn is used for quality estimation of a solution and derivation of an efficient termination criterion.

Furthermore, this dissertation presents an approach, which is able to run the proposed algorithms in a completely online fashion. Hereby, the main challenge is to integrate the heuristic into the execution of real-time tasks, which is solved by mapping iterations of the algorithm to hyper periods of the task execution. In doing so, a system becomes self-adaptive to dynamic changes. More importantly, it can be shown that the run-time overhead introduced by this approach is provably low.

# Zusammenfassung

Für moderne elektronische Systeme, insbesondere batteriebetriebene Geräte, spielt der Energieverbrauch eine immer wichtigere Rolle. Geringer Stromverbrauch und lange Akkulaufzeit sind die wichtigsten Anforderungen bei der Entwicklung, um die Betriebskosten der Geräte zu reduzieren. Auf Systemebene gibt es zwei weit verbreitete Techniken, um den Energieverbrauch zu reduzieren: Dynamic Power Management (DPM) und Dynamic Voltage and Frequency Scaling (DVS). Beide Techniken sind in der Lage, den Trade-off zwischen Systemleistung und Stromverbrauch zu regulieren. Während die DPM Technik versucht Systemkomponenten auszuschalten, wenn diese nicht benutzt werden, versucht die DVS Technik die Ausführungsgeschwindigkeit der Systemkomponenten zu verlangsamen. Da beide Techniken den Energieverbrauch auf Kosten der Systemleistung reduzieren, sollten sie insbesondere in der Kombination mit Echtzeitsystemen mit Bedacht eingesetzt werden.

Um den Energieverbrauch in Echtzeitsystemen zu reduzieren, beschäftigt sich diese Arbeit mit dem Problem der Energieverbrauchsoptimierung mit Hilfe einer kombinierten Anwendung von DPM und DVS. Dabei werden sowohl Einzelkernprozessor- als auch Mehrkernprozessorsysteme betrachtet. Hiermit wird insbesondere der Aufwand beim Zustandswechsel für DPM und DVS untersucht. Leider ist das betrachtete Optimierungsproblem NP-hart, sodass für seine Lösung keine effizienten Algorithmen existieren. Daher wird in dieser Dissertation ein heuristischer Suchalgorithmus entwickelt, der den Simulated Annealing um spezielle Regeln für die Selektion von Nachbarn erweitert. Darüber hinaus wird eine auf Regression basierte Technik zur Analyse des Verhaltens des vorgestellten Algorithmus erarbeitet. Aus dieser Technik wird zudem ein Abbruchkriterium des Algorithmus abgeleitet.

Ferner präsentiert diese Dissertation einen Ansatz zur Onlineausführung des vorgestellten Algorithmus. Dabei besteht die größte Herausforderung darin, dass der heuristische Algorithmus in der Ausführung des Echtzeitsystems integriert werden muss. Die Grundidee besteht darin, die Iterationen des Algorithmus auf die Hyperperiode der ausgeführten Echtzeitprozesse abzubilden. Dadurch ist das System in der Lage, sich selbstständig an dynamische Veränderungen anzupassen. Noch wichtiger ist jedoch der geführte Nachweis, dass der Laufzeitaufwand der Onlineausführung gering ist.

# Acknowledgement

This dissertation is the result of my research at C-LAB and University of Paderborn. I would like to take this opportunity to express my gratitude to several people, who supported me with their comments, suggestions and patience during the development of this work.

First of all, I would like to thank my supervisor Prof. Dr. Franz J. Rammig for his invaluable guidance and constructive feedback on the concepts I developed in this dissertation. I also thank Prof. Dr. Achim Rettberg for vice-supervising my dissertation. Furthermore, I would like to thank Prof. Dr. Sybille Hellebrand, Prof. Dr. Marco Platzner, and Prof. Dr. Christian Plessl for taking part in the examination board. I also thank Prof. Dr. Gregor Engels for his support at the final stage of writing this dissertation.

I am especially grateful to my group leader Dr. Wolfgang Müller for his advice and suggestions on finding my research topics.

This work would not have been possible without the fruitful discussions I had with my colleagues. My gratitude goes to Dr. Henning Zabel, Dr. Alexander Krupp, Tao Xie, Kay klobedanz, Jan Jatzkowski, Gilles Bertrand Gnokam Defo, Fabian Mischkalla, Markus Becker, Christoph Kuznik, Dr. Marcio Oliveira, Diana Riemer, and Mabel Joy.

Last but not least, I would like to thank my family for their patient suggestions and especially my wife Qian Liu for her continuous support.

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation

Due to the continuous advancements in technology, electronic devices nowadays are more and more powerful and fast. A modern smart phone has more computation power than all the computer devices used by NASA in 1969 to land a man on the moon and send him safely back to the earth [Mil12]. However, this enormous progress has its price, namely the power consumption of electronic systems is increasing rapidly as well. According to a report from Vodafone [FZ08], the power consumption in mobile communication systems rose 16-20% per year from 2003 to 2005. Another study [Lam+12] shows that the relative contribution of communication network electricity consumption to the total worldwide electricity consumption increased from 1.3% in 2007 to 1.8% in 2012. Therefore, energy efficiency becomes one of the most important requirements in the system development to make the products more competitive in the market.

Figure 1.1 shows the power consumption and performance trend of several latest ARM application processors, which are increasing exponentially following Moore's law [Moo98]. There is no doubt that this dramatic growth of power consumption presents a really big challenge in the system design, especially for those with a constrained power budget. However, the good news is that the power consumption shown here is the maximal value, i.e., the power consumed by a processor when it is fully utilized. Fortunately, modern processors are usually equipped with run-time energy management techniques, which are able to dynamically reduce the system energy consumption. Dynamic Power Management (DPM) and Dynamic Voltage and Frequency Scaling (DVS or DVFS) are two mostly applied techniques, where DPM tries to shut down an unused component and DVS tries to slow down the operating speed of a working component. In general, both techniques reduce the energy consumption at the cost of performance degradation, which is a crucial point in the context of hard real-time systems. Real-time com-

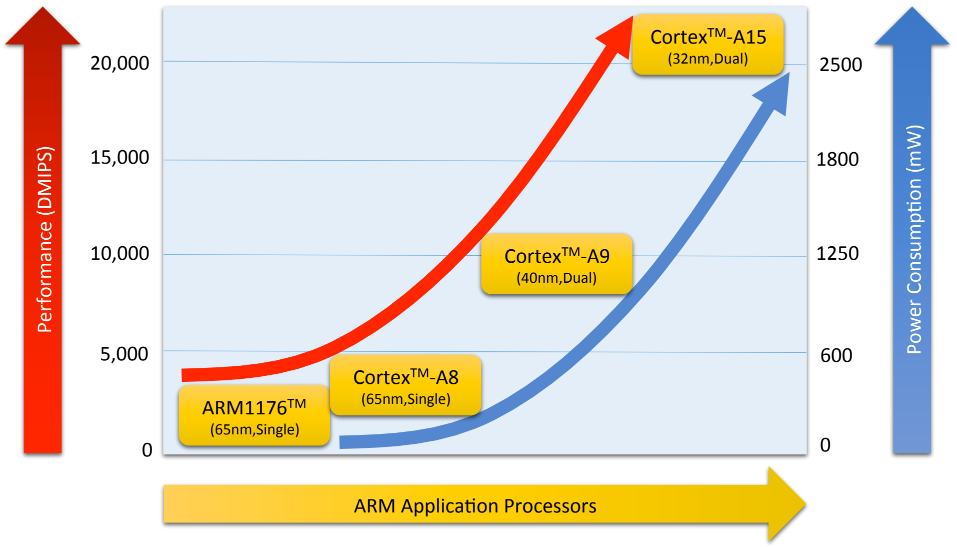


Figure 1.1: The trend of the power consumption vs. performance of ARM application processors [Ltd] [ITP] [Shi]

puting in computer science describes a special form of computation that is closely related to time. The computation result contains not only logic values, but also temporal properties. Typically, a real-time system is composed of a set of real-time tasks, which are defined as software programs that must complete before a given deadline. The execution of tasks does not need to be as fast as possible. It only has to hold timing constraints. In this context, a late answer is usually considered as incorrect. Depending on the consequences caused by a deadline miss, there are three different categories of real-time tasks: hard real-time, firm real-time and soft real-time. This dissertation focus on the hard real-time property, where timing constraint violations are strictly prohibited.

In fact, real-time applications are becoming increasingly important and ubiquitous in daily life. As a common feature, they often have to interact with environment and therefore in most cases are embedded into computing systems. For instance, an airbag control system in vehicles is a typical hard real-time embedded system, since a delayed response may cause damage. Many real-time embedded systems are operating under mobile condition. Some notable examples are telecommunication systems, robotics, portable consumer electronic devices and space applications. In such systems, power consumption clearly is of utmost importance. In order to keep energy consumption manageable, both DPM and DVS techniques are widely applied in this context. However, due to the timing constraints, they have to be used with great caution.

This dissertation exactly deals with this challenge by finding the best way to apply the DPM and DVS techniques in hard real-time systems. In the following sections, the main objectives and challenges are described. Thereafter, the key contributions of the dissertation are highlighted.

## 1.2 Objectives

In short, the main objective of the dissertation is to optimize the total energy consumption in hard real-time systems by taking the advantage of DPM and DVS. Hereby there are several important requirements.

**System schedulability**: As hard real-time systems are addressed, the timing constraints must be satisfied. A real-time system is said to be schedulable or feasible, if all the given tasks can finish the execution before their deadlines.

**System-wide energy minimization**: Usually, a real-time system is composed of software and hardware. The hardware in turn contains multiple components, including the processor and I/O devices. The goal here is to not only optimize the processor energy consumption, but also the energy consumption of an entire system.

**Consideration of state switching overhead**: Both DPM and DVS techniques reduce energy consumption by means of changing the operation mode of components. An operation mode is also referred to as a power state. As a state switching incurs time and energy overhead, they need to be carefully dealt with.

**Online**: In the context of energy optimization in real-time systems, there are online and offline approaches. Clearly, online approaches are more advanced in terms of flexibility, because they are adaptive to system changes. However, a big challenge is that the run-time overhead needs to be as low as possible. One of the objectives of the dissertation is that the proposed approach should be online and adaptive.

**Consideration of multi-core processors**: As multi-core processors become more and more popular, the energy aware real-time scheduling problem on such platforms needs to be addressed.

## 1.3 Problem Statement

By optimizing the system-wide energy consumption in hard real-time systems, there arise several problems and challenges.

**DPM usage**: The main idea behind the DPM technique is to switch off unused system components to save energy. For instance, a processor may support several operation states. One of them is the `run` state, in which the processor executes instructions. The other states are `standby` and `sleep`. Whenever the processor becomes idle, it can be switched to the `standby` or `sleep` state. However, the state switching usually causes both time and power penalty and can not be mindlessly ignored, especially when task execution time is in the same order of magnitude of switching latency. The deeper the sleep state, the more penalty has to be paid. Thus it takes longer to enter the `sleep` state than the `standby` state. In fact, due to the non-negligible overhead, a component can only be switched off under particular conditions. More concretely, a component shutdown is dependent on the length of idle time. Only if the idle time is long enough, it is worth switching off the component. By analogy, nobody will suspend a computer to save energy while writing a document, even when occasionally the user needs to stop typing and think about the content, because the idle time is simply too short. In the context of hard real-time systems, an unjustified component shutdown not only causes additional energy consumption, it may even delay upcoming work and thus jeopardize deadlines. Therefore the main challenge by using the DPM technique is to predict idle time and decide whether a component could be switched off, and if so, to which sleep state.

**DVS usage**: The basic concept of the DVS technique is to slow down the component operating speed or frequency. In general, the power consumption of a CMOS based circuit is quadratically proportional to the supply voltage. Reducing the voltage and thus the speed, dramatically decreases the power consumption. However, since running a task at lower speed will increase the execution time, it should be carefully used, so that no task misses its deadline. Therefore, the main problem is to find the most appropriate operating speed for each task to minimize the system energy consumption while meeting all the timing constraints.

**Combined usage of DPM and DVS**: If the DPM and DVS techniques are used together, they actually work contradictorily. More concretely, on one hand, the DPM technique tries to complete the work as soon as possible, so that more idle time is available for DPM application. On the other hand, the DVS tries to use the lowest speed and thus complete the work as late as possible. The extensive usage of the DPM technique will limit the possibility of DVS usage and vice versa. The main challenge hereby is to find the best compromise or trade-off between them.

**DPM/DVS usage on multi-core platforms**: Multi-core processors are coming into market in modern electric devices. In this context, the energy optimization problem for hard real-time systems becomes even more complex, as an energy efficient task partition should be additionally derived. Moreover, there exist some hardware constraints on multi-core processor platforms in terms of the DPM and DVS capabilities. For instance, if some processor cores share the same power supply net, then they can only operate at the same speed at the same time. An energy efficient real-time scheduling needs to take this into consideration.

## 1.4 Contributions

In order to achieve the objectives, this dissertation makes the following contributions.

**System model**: The first contribution is proposing an abstract system model and formally formulating the main problem. Hereby, the idea of power state machine is adopted to describe the power characteristics of processors and devices. For processors, this dissertation addresses both single-core processor platforms and multi-core processor platforms. Moreover, the traditional real-time task model is adopted to represent software applications.

**Heuristic Search Algorithm**: The second contribution is proposing a heuristic search algorithm based on simulated annealing to solve the energy optimization problem. The traditional simulated annealing algorithm is extended by incorporating domain specific information, which improves the algorithm performance. In other words, rules are defined to guide the search of neighbor solutions.

**Run-Time Behavior Analysis**: The third contribution is analyzing the run-time behavior of the heuristic search algorithm. A run-time mechanism based on the exponential regression technique is introduced to estimate system performance. An efficient and accurate termination criterion for the heuristic search algorithm is derived as well.

**ES-AS Approach**: The fourth contribution is proposing an online approach, which runs the heuristic search algorithm in a completely adaptive fashion. Hereby the main idea is to integrate the iterative algorithm into real-time system execution. The online approach is divided into two stages, the Exploration Stage (ES) and the Application Stage (AS). This is the reason why it is called ES-AS approach. In short, the exploration stage tries to explore candidate solutions, and the best solution found is applied in the application stage. As the approach is completely online, the information concerning system dynamics can be easily considered. Moreover, the run-time overhead introduced by the approach, which will be discussed in detail later, is provably small.

**DVS state switching overhead handling**: Unlike the DPM state switching overhead, which has attracted a lot of attention in existing work, the DVS state switching overhead is often ignored, though it incurs considerable overhead as well. The fifth contribution of this dissertation is investigating the impact of the non-negligible DVS state switching overhead and the necessity of its treatment. Two run-time protocols, conservative protocol and speed inheritance protocol, are proposed to solve the problem.

## 1.5 Organization of the Dissertation

The remainder of the dissertation is organized as follows. Chapter 2 gives the fundamentals and backgrounds in the area of energy efficient real-time scheduling. An overview of the state-of-the-art analysis is shown as well. Chapter 3 introduces the system model and formally defines the optimization problem for both single-core and multi-core processor platforms. Chapter 4 presents the details of the heuristic search algorithm and the subsequent Chapter 5 analyzes its run-time behavior. A termination criterion of the heuristic search algorithm is also derived. Chapter 6 proposes the online approach and proves its correctness in terms of system schedulability and efficiency in terms of run-time overhead. In order to deal with the non-negligible DVS state switching overhead, Chapter 7 investigates its influence and proposes two run-time protocols called conservative protocol and speed inheritance protocol. Before the dissertation is finally concluded in Chapter 9, a thorough evaluation of the proposed algorithm and approach is performed and given in Chapter 8.

# Chapter 2

# Background

This chapter gives the fundamentals of the dissertation. The first section introduces the basic concept behind real-time systems and reviews some important scheduling algorithms. The second section addresses the background in the area of energy management techniques with special focus on run-time aspects. In the subsequent section, a state-of-the-art analysis is presented concerning the energy optimization problem in hard real-time systems. Before this chapter is finally concluded, the Advanced Configuration & Power Interface standard is introduced.

## 2.1 Real-Time Systems and Scheduling

Real-time computing in computer science is a special term used to describe the computation, where correctness is not only dependent on the logical value but on the temporal constraints as well [Kop11]. A real-time system must be able to produce correct results on time or earlier. A late answer is considered to be wrong.

In fact, there exists a wide range of applications that require the real-time property. Some examples include automotive applications, telecommunication systems, robotics, multimedia systems, space missions, consumer electronic devices, etc. [But11] [Mar03]. In general, most real-time systems are applied in the embedded context, i.e., encapsulated in an environment and designed for a specific and dedicated purpose. Because of the interaction with environment, one of the most important characteristics in real-time systems is being reactive. Hereby time clearly plays a crucial role. One common misunderstanding of real-time computing is often associated with ”being fast”. However, the term ”fast” is only relative to the environment. Instead, the real-time requirement does not mean running as fast as possible, but rather completing the computation within a predefined deadline. In other words, the real-time property is more related to predictability and determinism of systems.

In terms of safety critical systems, real-time constraints can be classified in three categories depending on the consequence of a possible deadline miss [But11]:

- Hard Real-Time: A system is said to be a hard real-time system, if a deadline violation may result in catastrophic consequences.
- Firm Real-Time: A system is said to be a firm real-time system, if produced results are useless but will not cause any damage when they are late.
- Soft Real-Time: A system is said to be a soft real-time system, if a late answer only reduces system performance but is still useful, at least to some extend.

A real-time system is usually composed of hardware and software, which in turn comprises a set of processes. From the system point of view, a software process is considered as a basic computation entity executed by the processor. In the context of real-time systems, a software process is commonly referred to as a task. Typically, a real-time system contains more than one task and thus admits a multitask execution environment. Since the tasks might be concurrent, their execution order needs to be decided by a so-called scheduling algorithm. In other words, the scheduling algorithm is in charge of dispatching all the tasks at the right time. Moreover, a task is said to be preemptive, if its execution can be interrupted by another task with higher priority. Otherwise, the task is non-preemptive. Conventionally, a set of real-time tasks is often denoted by $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$, where each task $\tau_i$ is characterized by the following properties [But11]:

- **Release time** is the time, at which the task is released and becomes ready for execution.
- **Start time** is the time at which the task actually starts its execution.
- **Worst Case Execution Time (WCET)** is the time needed by the task to complete the execution without preemption under the assumption of worst case conditions.
- **Actual Execution Time (AET)** is the actual time needed by the task to complete the execution without preemption.
- **Completion time** is the time at which the execution of the task is finished.

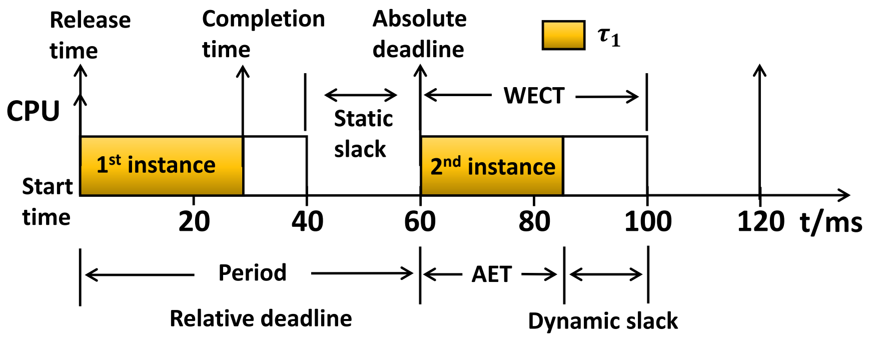


Figure 2.1: A real-time task with its properties

- **Absolute deadline** is the time, before which the task must finish its execution.
- **Relative deadline** is the difference between the absolute deadline and the release time of the task.
- **Static slack** is the maximum time a task may be procrastinated without violating the deadline based on its WCET.
- **Dynamic slack** is the time coming from the task earlier completion than the WCET, i.e., the difference between WCET and AET.

In many real-time systems, it is quite common that the tasks are periodic, i.e., the execution will be repeated at a specific rate. To distinguish task executions in different periods, the definition of the so-called job instance is introduced. The $j$-th job instance of a task identifies the task execution in the $j$-th period. As a common assumption, the repetition rate of tasks is considered as a constant.

Figure 2.1 illustrates the above mentioned properties based on a simple example with one real-time task. The task $\tau_1$ is released at 0 ms and the relative deadline is 60 ms, which is equal to the period. Moreover, the task WCET is assumed to be 40 ms.

In terms of task dependencies, real-time systems can be distinguished into two categories. The first category contains such task sets, in which the tasks are dependent upon each other. More specifically, the tasks have to be executed in a predefined order according to their precedence relation. Usually, these tasks together with their dependencies can be described as a directed acyclic graph. On the contrary, the second type of real-time systems contains independent tasks, which can be executed in arbitrary order. For the sake of simplicity, this dissertation concentrates on independent tasks only. However, the possibility of handling tasks with precedence relation is discussed in the conclusion chapter.

In order to successfully execute all the tasks without missing any deadline, real-time scheduling algorithms play a key role. Formally, a schedule on a single-core processor is defined as a total order on a set of real-time tasks by deciding the start time of each instance of the tasks. A real-time system is said to be schedulable or feasible, if there exists a real-time schedule such that all the tasks are able to complete their respective execution before their deadlines. Real-time scheduling analysis is a well-studied research area. There exist a large variety of real-time scheduling algorithms. A comprehensive review of such algorithms is out of the scope of this dissertation. More detailed information can be found in [But11]. In what follows, the main focus is put on the two well-known real-time scheduling algorithms, Earliest Deadline First (EDF) and Rate Monotonic (RM), which are widely applied for periodic tasks.

Before the EDF and RM scheduling algorithms are explained, the processor utilization $U$ will be first introduced. Given a set of real-time tasks, $U$ is defined as the portion of time, in which the processor executes tasks. Formally, it is computed by

$$U = \sum_{i=1}^{n} \frac{W(\tau_i)}{T(\tau_i)} \tag{2.1}$$

where $W(\tau_i)$ and $T(\tau_i)$ denote the WCET and the period of the task $\tau_i$, respectively.

The EDF scheduling algorithm is a dynamic approach, which dispatches tasks according to run-time parameters. At each time, among all the ready tasks the one with the earliest absolute deadline will be executed. In fact, each task is associated with a priority based on its absolute deadline. The earlier the deadline, the higher priority a task has. Thus, the priority of tasks may change at run-time. By definition, the EDF algorithm is preemptive. There exists a schedulability analysis based on the processor utilization factor, which yields a sufficient and necessary condition for EDF to guarantee system feasibility. In particular, Theorem 2.1.1 holds under the following assumptions:

- A1: For all the tasks, the period is constant.
- A2: The WCET of all the job instances of a periodic task is constant.
- A3: The relative deadline of each task is equal to its period.
- A4: All the tasks are independent.

**Theorem 2.1.1** (Theorem 4.2 in [But11]). *A set of periodic tasks is schedulable with EDF if and only if the processor utilization*

$$U \leq 1. \tag{2.2}$$

*Proof.* The proof can be found in [But11]. □

The RM scheduling algorithm is a static approach, where tasks are dispatched according to some fixed parameters that will not change at run-time. More concretely, each task is associated with a static priority based on its period, i.e., the longer the period, the lower priority a task has. At run-time, always the task with highest priority is selected for execution. Instinctively, the RM scheduling algorithm is preemptive. For schedulability analysis, Theorem 2.1.2 gives a sufficient condition based on the processor utilization test under the assumptions A1, A2, A3 and A4.

**Theorem 2.1.2.** *Given n real-time periodic tasks, the system is schedulable with RM if the processor utilization*

$$U \leq n(2^{1/n} - 1) \tag{2.3}$$

*Proof.* [LL73] has shown that the least upper bound to the processor utilization for the RM algorithm is $n(2^{1/n} - 1)$. By definition [LL73], if the processor utilization is less than the least upper bound, then all the tasks are schedulable. Thus, the theorem follows directly. □

For $n \geq 1$, $n(2^{1/n} - 1)$ is a strictly decreasing function of $n$. As $n$ grows to infinity, $n(2^{1/n} - 1)$ eventually converges to $\ln 2 \approx 0.69$ [But11]. Thus the schedulability test can be simplified to

$$U \leq 0.69 \tag{2.4}$$

Clearly, the schedulability tests for both EDF and RM have a linear computation complexity $O(n)$.

Figure 2.2 illustrates the different task execution of an example by applying the EDF and RM algorithms. The example consists of two real-time tasks. The WCET and period/deadline of $\tau_1$ is 5 ms and 20 ms, respectively. The WCET and period/deadline of $\tau_2$ is 17.5 ms and 30 ms, respectively. The hyper period of a task set is defined as the least common multiple of all the task periods, which is 60 ms in this example. The task execution repeats

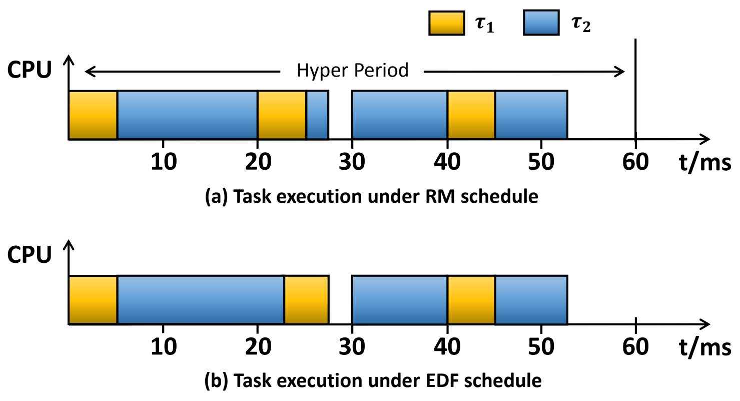


Figure 2.2: Different task execution under EDF and RM

itself in each hyper period. The main difference between the schedules applying EDF and RM happens at 20 ms when $\tau_2$ becomes active. In case of the RM algorithm, $\tau_1$ has a higher priority than $\tau_2$, because its period is shorter. Therefore, the execution of $\tau_2$ is interrupted by $\tau_1$. However, in case of the EDF scheduling algorithm, $\tau_2$ has a higher priority, because the absolute deadline of its current instance is due at 30 ms, which is earlier than the absolute deadline of the current instance of $\tau_2$. Therefore, the execution of $\tau_2$ is not preempted by $\tau_1$.

Until now this section only addressed single-core processor based real-time systems. In what follows, a brief overview of real-time scheduling on a multicore processor is given. As will be explained later, this work focuses on independent real-time tasks. In this context, there exist two classes of real-time scheduling algorithms: the partitioned scheduling and the global scheduling [DB11]. The key difference between them is that the task migration is not allowed in the former case whereas it is permitted in the latter case. In other words, the partitioned approach tries to allocate processor cores to tasks beforehand and later at run-time a task may only run on its allocated processor core. Global scheduling, on the other hand, may dynamically change the processor core allocation at run-time if needed, i.e., different job instances of a task may be executed on different processor cores. One of the well-known scheduling algorithms in this category is the Proportionate Fair (Pfair) algorithm, which breaks the time into slots with equal length and each time slot is allocated to a task. A detailed review of global scheduling algorithms can be found in [DB11].

This dissertation concentrates on the partitioned scheduling, because it provides several benefits in practice compared to the global scheduling.

- There is no penalty overhead for task migration.
- If a task has missed the deadline, only the tasks on the same processor core are affected.

- The partitioned scheduling is more efficient in terms of scheduling overhead, because each processor core only has to deal with the tasks partitioned on it. On the contrary, the global scheduling has to manage a global queue for all the tasks and often make the scheduling decision online.

- Once the tasks are partitioned, the well-known single-core real-time scheduling algorithms, such as EDF and RM, can be applied for each processor core without adaptation.

Due to the last point shown above, the main challenge of real-time scheduling on a multi-core processor obviously is reduced to a task partition problem. Fortunately, it can be transformed to the bin packing problem [Vaz02], a well-studied combinatorial optimization problem, where the main objective is to pack $n$ objects with different values into a set of bins with different capacities while minimizing the number of bins. Clearly, the tasks represent the objects and the object value is the utilization of the task, i.e., $W(\tau_i)/T(\tau_i)$. The processor cores represent the bins and the bin capacity is the utilization upper bound on the processor core to ensure system schedulability based on a particular real-time scheduling algorithm, i.e., the capacity is 1 or 0.69 for the EDF or RM scheduling algorithm, respectively. However, the goal in the scheduling context is not to minimize the number of applied processor cores, but rather to partition tasks onto a set of given processor cores, so that all the tasks are schedulable. Other optimization goals, such as minimizing power consumption or response time, can be added to the problem.

A comprehensive review of the partitioned scheduling is provided in [DB11]. Because the bin packing problem is proven to be $\mathcal{NP}$-hard, most studies try to use heuristics to partition tasks. Some common strategies are listed as follows.

- **First Fit (FF)** strategy tries to partition a task to the first processor core, which still has enough space for it.

- **Next Fit (NF)** strategy tries to partition a task to the next processor core (with the next higher index), which still has enough space for it.

- **Best Fit (BF)** strategy tries to partition a task to the mostly loaded processor core (i.e., with highest utilization), which still has enough space for it.

- **Worst Fit (WF)** strategy tries to partition a task to the least loaded processor core (i.e., with lowest utilization), which still has enough space for it.

Once the tasks are successfully partitioned, single-core processor real-time scheduling algorithms can be applied on the processor cores. In order to run the partitioned tasks on different processor cores independently, the partitioned scheduling needs to manage a separate task queue per processor core.

## 2.2 Run-Time Energy Management Techniques

In a CMOS-based integrated circuit, power consumption is mainly composed of two parts: the dynamic power consumption $P_{dynamic}$ and static power consumption $P_{static}$.

$$P = P_{dynamic} + P_{static} \tag{2.5}$$

The dynamic power dissipation comes from charging and discharging processes of the logic circuit and therefore is highly dependent on the switching activity and speed. In general, it can be expressed by (2.6) [ZMC03],

$$P_{dynamic} = C_{ef} \cdot V_{dd}^2 \cdot F \tag{2.6}$$

where $C_{ef}$ is the effective switching capacitance of the integrated circuit, $V_{dd}$ is the supply voltage and $F$ is the clock frequency or speed. Moreover, the clock frequency is dependent on the supply voltage

$$F = k \cdot \frac{(V_{dd} - V_{th})^2}{V_{dd}} \tag{2.7}$$

where $k$ is a technology constant and $V_{th}$ is the threshold voltage [ZMC03]. By assuming that the threshold voltage is relatively small, the clock frequency can be approximated as a linear function of the supply voltage.

$$F \approx k \cdot V_{dd} \tag{2.8}$$

By substituting $V_{dd}$ in (2.6) with $\frac{F}{k}$, the dynamic power consumption can be considered as a cubic function of the clock frequency [ZMC03].

$$P_{dynamic} = P_{dynamic}(F) \approx \frac{C_{ef}}{k^2} \cdot F^3 \tag{2.9}$$

The static power dissipation mainly describes the leakage power consumed by transistors disregarding whether there are charging or discharging activities. Typically it can be expressed by

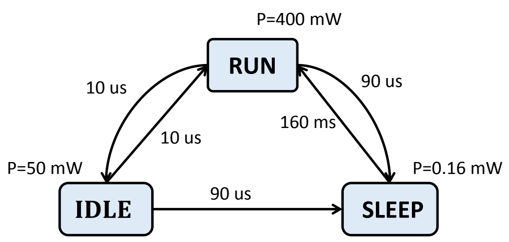


Figure 2.3: Power state machine of the StrongARM SA-1100 processor [BBM00]

$$P_{static} = I_{static} \cdot V_{dd} \tag{2.10}$$

where $I_{static}$ is the total current flowing in the circuit. In early years, the static power consumption was often ignored, because it was clearly dominated by the dynamic power consumption. However, due to the rapid and continuous advancement toward nanoscale technology in deep sub-micron regimes, the gate size becomes smaller and smaller. As a result, the leakage current and thus the static power consumption becomes larger and larger [Aga+06]. This leads to an increasing need to efficiently manage and deal with the static power consumption.

As mentioned in the introduction chapter, there are two major run-time energy management techniques from the system level point of view: the DPM and DVS techniques. In general, the DPM technique is applied to reduce the static part of power consumption while the DVS technique is primarily used to save the dynamic power consumption. In what follows, both techniques are discussed in detail.

### 2.2.1 Dynamic Power Management (DPM)

Generally, ”dynamic power management” is a frequently used term in the research area of low power system design. There are many different interpretations. To avoid ambiguity, in this dissertation Dynamic Power Management (DPM) is particularly referred to as a technique that dynamically adjusts system power consumption by selectively switching off components when they are idle. The primary motivation behind this technique is to reduce static power consumption. A system component with DPM capability is commonly designated as a power manageable component [BBM00], which can be modeled by a power state machine. From the system point of view, a component can be a processor or an I/O device, such as ethernet controller, hard disk, memory card driver, USB controller etc.. Figure 2.3 illustrates the power state machine of an example processor, StrongARM SA-1100 [BBM00]. Hereby the `RUN` state is the only working state, in which

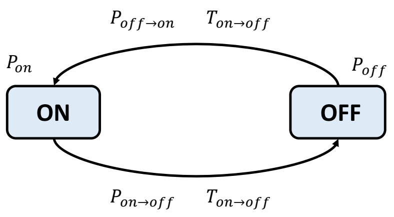


Figure 2.4: Power state machine of a component with two states [BBM00]

the processor may execute software processes. The other two states `IDLE` and `SLEEP` are referred to as low power states, also known as sleep states, which can be used for power saving. The `RUN` state consumes the most power (400 mW) and the `SLEEP` state consumes the least power (0.16 mW). Moreover, as shown in Figure 2.3, the state switching incurs time overhead. Though the latency appears to be very small, mindless ignorance is not always justified, especially if the task execution time is in the same order of magnitude.

The main cause of the switching overhead is accounted to

- switching on/off and stabilizing the power supply.
- stabilizing the clock.
- loading and storing the system context.

Obviously, the deeper the low power state, the more power can be saved, however the higher switching overhead has to be paid. There is a trade-off by choosing the most appropriate low power state between power saving and switching overhead. In some cases, the component should not even enter a low power state at all, because the induced overhead is higher than the saved power. In order to deal with this problem, the concept of so-called break-even time [BBM00] is adopted. To simplify the explanation, the discussion is based on a simple example shown in Figure 2.4.

The example component supports two states, `ON` and `OFF`, indicating the active and the sleep state, respectively. $P_{on}$ and $P_{off}$ denote their respective power consumption. As mentioned above, a state switching introduces non-negligible overhead. Thus, $P_{on\rightarrow off}$ and $T_{on\rightarrow off}$ denote the power consumption and the latency of the state switching from `ON` to `OFF`. Similarly, $P_{off\rightarrow on}$ and $T_{off\rightarrow on}$ denote the power consumption and the latency when switching the state from `OFF` back to `ON`.

Considering a scenario that the component becomes idle at a certain time $t_1$ and will be required later again at $t_2$, i.e., the idle time is $[t_1, t_2]$. Clearly, there are two possibilities concerning the usage of DPM.

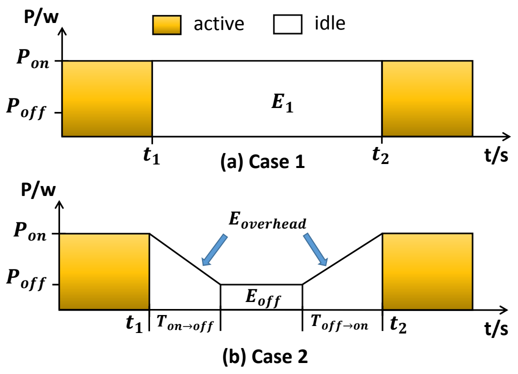


Figure 2.5: Comparison of energy consumption with and without component shutdown

- **Case 1**: The DPM technique is not applied at all. The component remains in the state `ON` and thus no state switching overhead needs to be considered. The total energy $E_1$ consumed in the interval $[t_1, t_2]$ can be computed in (2.11). Figure 2.5(a) illustrates this situation.

$$E_1 = P_{on} \cdot (t_2 - t_1) \tag{2.11}$$

- **Case 2**: Figure 2.5(b) shows the case, where the component is switched off to the state `OFF` during $[t_1, t_2]$. In order not to delay the upcoming work, the component needs to be switched on before its actually required time $t_2$. Thus the inequation (2.12) expresses a condition for this case.

$$t_2 - t_1 \geq T_{on\to off} + T_{off\to on} \tag{2.12}$$

By incorporating the switching overhead, the total energy consumption $E_2$ in $[t_1, t_2]$ can be computed in (2.13).

$$E_2 = E_{off} + E_{overhead} \tag{2.13}$$

where $E_{off}$ denotes the energy consumed when the component is in the `OFF` state.

$$E_{off} = P_{off} \cdot (t_2 - t_1 - T_{on\to off} - T_{off\to on}) \tag{2.14}$$

and $E_{overhead}$ denotes the energy consumption overhead caused by the state switching.

$$E_{overhead} = P_{on\to off} \cdot T_{on\to off} + P_{off\to on} \cdot T_{off\to on} \tag{2.15}$$

Clearly, Case 2 is more favorable in terms of energy saving, if $E_1 \geq E_2$. Otherwise Case 1 is more preferred. Thus by substituting $E_1$ and $E_2$ with the expressions from (2.11) and (2.13), the condition to choose Case 2 can be obtained.

$$P_{on} \cdot (t_2 - t_1) \geq E_{off} + E_{overhead} \tag{2.16}$$

By applying (2.14), (2.16) can be reformulated to (2.17).

$$\begin{aligned} P_{on} \cdot (t_2 - t_1) &\geq P_{off} \cdot (t_2 - t_1 - T_{on \to off} - T_{off \to on}) + E_{overhead} \\ &\geq P_{off} \cdot (t_2 - t_1) - P_{off} \cdot (T_{on \to off} + T_{off \to on}) + E_{overhead} \end{aligned} \tag{2.17}$$

If (2.17) is reformulated by combining the term $t_2 - t_1$ on both sides of the equation, (2.18) can be obtained.

$$(P_{on} - P_{off}) \cdot (t_2 - t_1) \geq E_{overhead} - P_{off} \cdot (T_{on \to off} + T_{off \to on}) \tag{2.18}$$

By further reformulation, the condition for Case 2 can be expressed by (2.19).

$$t_2 - t_1 \geq \frac{E_{overhead} - P_{off} \cdot (T_{on \to off} + T_{off \to on})}{P_{on} - P_{off}} \tag{2.19}$$

Intuitively, there is a lower bound to the idle time. Only if the idle time lasts longer than it, switching off the component is beneficial. Otherwise, the energy consumption overhead caused by the state switching is so high, that it would be better to keep the component in the `ON` state. Looking at (2.14), the longer the idle time, the more energy can be saved, i.e., the more benefit can be achieved by switching off the component. Thus the break-even time is defined as the minimum time required for a component being idle, so that the switching overhead can be compensated. In other words, only if the saved energy exceeds the overhead, it makes sense to choose Case 2, i.e., enter the `OFF` state. Note that the condition (2.12) is another prerequisite for Case 2, i.e., it presents another strict lower bound for $t_2 - t_1$. Combining the conditions (2.12) and (2.19), the break-even time for the `OFF` state, denoted by $t_{be,off}$, is obtained.

$$t_{be,off} = \max\{T_{on \to off} + T_{off \to on}, \frac{E_{overhead} - P_{off} \cdot (T_{on \to off} + T_{off \to on})}{P_{on} - P_{off}}\} \tag{2.20}$$

Since each low power state has its own switching overhead, different low power states have different break-even time. Obviously, the deeper the low power state, the less power it consumes, however, the longer the break-even time takes. If a component supports multiple low power states, there is a question concerning how to select the most appropriate one, if it becomes idle. In order to maximize power saving, the chosen state should be as low as possible, i.e., the deepest low power state, whose break-even time is yet longer than the idle time.

As a summary, due to non-negligible DPM state switching overhead, a component can not be simply switched off whenever it becomes idle. The decision is rather dependent on the length of idle time. Only if the idle time is longer than the corresponding break-even time, the component may enter a low power state. An unjustified shutdown not only causes more energy, but sometimes even delays upcoming work, which is a crucial point in hard real-time systems.

### 2.2.2 Dynamic Voltage and Frequency Scaling (DVS)

Another well-applied run-time energy management technique is the so-called Dynamic Voltage and Frequency Scaling (DVS), also known as DVFS. As opposed to the DPM technique, the main motivation behind DVS is to reduce the dynamic power consumption of circuits by slowing down the current operating frequency/speed and the corresponding supply voltage. The relationship between operating speed and supply voltage can be roughly modeled as a linear function (2.8) [ZMC03]. In general, a pair of supply voltage and frequency is referred to as an operation point. The DVS technique is primarily available on processors, such as the well-known Intel SpeedStep® technology. Note that this dissertation is not interested in reducing operating frequency while keeping supply voltage, because it is not beneficial from the energy saving point of view. However, it can be applied to reduce system peak power consumption or for thermal control purpose, but this would be out of the scope of this work.

To simplify explanation and avoid ambiguity in the remaining text, if the term ”speed” or ”frequency” is used in the context of being increased or decreased, operating voltage is always implicitly assumed to be scaled accordingly as well.

As indicated in (2.9), dynamic power consumption can be approximated as a cubic function of speed. Therefore, reducing operating frequency results in a cubic decrease of power consumption. However, this power reduction comes at a price, because the execution time of workload is extended. There is a trade-off between performance and power consumption. Especially in the context of hard real-time systems, the DVS technique has to be applied with great caution, as no task should ever violate its timing constraints. Furthermore, this work assumes that the task WCET is a linear decreasing function of operating speed, i.e., the WCET decreases linearly as processor speed increases.

In order to analyze the DVS technique, the so-called Active Energy Consumption of a task is defined.

**Definition 2.2.1.** *The **Active Energy Consumption (AEC)** of a periodic task $\tau_i$, denoted by $E_{active}(\tau_i, F)$, is defined as the energy consumed during the task execution at a particular frequency $F$ within one period.*

Intuitively, a task AEC describes the energy consumed by the task when it is active. Formally, it can be computed as follows.

$$E_{active}(\tau_i, F) = P(F) \cdot \frac{W(\tau_i) \cdot F_{ref}}{F} \tag{2.21}$$

where $P(F)$ is the processor power consumption when running at the speed $F$. Since the task execution time varies along with the operating frequency, $W(\tau_i)$ denotes the WCET of $\tau_i$ based on a reference frequency $F_{ref}$. Without loss of generality, $F_{ref}$ is assumed to be the highest speed and all the remaining speeds are lower. Obviously, $\frac{W(\tau_i) \cdot F_{ref}}{F}$ gives then the WCET of $\tau_i$ running at $F$. If only considering the dynamic power consumption, (2.21) can be refined by applying (2.9) as follows.

$$E_{active}(\tau_i, F) = P_{dynamic}(F) \cdot \frac{W(\tau_i) \cdot F_{ref}}{F} = \frac{C_{ef} W(\tau_i) F_{ref}}{k^2} \cdot F^2 \tag{2.22}$$

Clearly, $E_{active}(\tau_i, F)$ is a strictly increasing function of $F$. In order to minimize the task AEC, the operating frequency needs to be lowered as much as possible, i.e., using the lowest speed that can complete execution within the task deadline.

However, if the static power consumption is considered as well, the formula of $E_{active}(\tau_i, F)$ looks like as follows.

$$E_{active}(\tau_i, F) = \frac{C_{ef} W(\tau_i) F_{ref}}{k^2} \cdot F^2 + P_{static} \cdot \frac{W(\tau_i) F_{ref}}{F} \tag{2.23}$$

For a certain task and a certain technology, $C_{ef}$, $k$, $P_{static}$, $F_{ref}$ and $W(\tau_i)$ are clearly constant. It is not hard to see that $E_{active}(\tau_i, F)$ is not a strictly increasing function of $F$ any more. In fact, it becomes merely a convex function, which admits a minimum value, i.e., the optimal energy consumption.

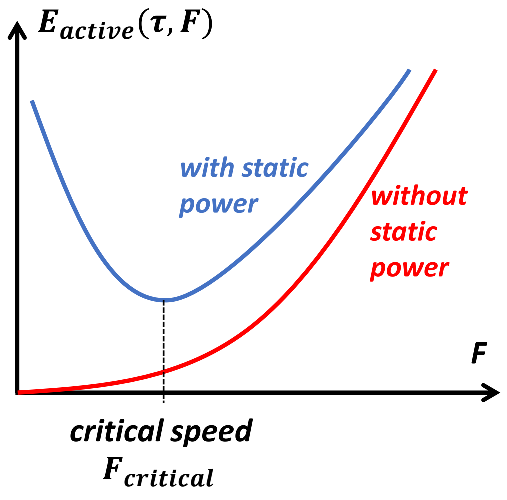


Figure 2.6: The curves of the task active energy consumption

In order to find the minimum, the derivative of the function needs to be determined.

$$\frac{dE_{active}(\tau_i, F)}{dF} = \frac{2C_{ef}W(\tau_i)F_{ref}}{k^2} \cdot F - P_{static} \cdot \frac{W(\tau_i)F_{ref}}{F^2} \qquad (2.24)$$

By letting $\frac{dE_{active}(\tau_i,F)}{dF} = 0$, the optimum energy consumption is obtained, if the task runs at the speed $F_{critical}$.

$$F_{critical} = \sqrt[3]{\frac{P_{static}k^2}{2C_{ef}}} \qquad (2.25)$$

In this context, $F_{critical}$ is commonly referred to as the critical speed or optimal speed. No task should ever run below this speed, because otherwise both the task AEC and execution time increase. Figure 2.6 illustrates the curves of the task AEC. The red and blue curves correspond to the functions in (2.22) and (2.23), respectively. Note that the critical speed may not be always suitable for the task, because it might be too slow to complete the task on time. It only gives a lower bound to choose the optimal task operating speed concerning its AEC. Considering the speeds above the critical speed, the task AEC becomes a strictly increasing function of speed again. Moreover, as shown in (2.25), $F_{critical}$ is independent of tasks and only related to power characteristics of the processor.

From the theoretical point of view, the formula (2.25) presents a rigorous analysis of the critical speed. However, some of the constants, such as $C_{ef}$, $k$ and $P_{static}$ are not always available. Therefore it is rarely used in practice. By definition, the critical speed is the optimal speed to run tasks (without

| Speed $F$ (MHz) | 150 | 400 | 600 | 800 | 1000 |
|---|---|---|---|---|---|
| Power consumption (mW) | 80 | 170 | 400 | 900 | 1600 |
| $W(\tau)$ (ms) | 667 | 250 | 167 | 125 | 100 |
| $E_{active}(\tau, F)$ (mJ) | 53.36 | 42.5 | 66.8 | 112.5 | 160 |

Table 2.1: An example to compute the critical speed [Xu+04]

consideration of deadlines) in terms of AEC. Since the number of available speeds of a processor is usually limited, the critical speed can be computed by first selecting one arbitrary task, then determining its AEC for all the speeds and finally choosing the speed with the minimal energy consumption. Table 2.1 shows an example, where the Intel XScale processor [Xu+04] supports 5 operating speeds (150 MHz, 400 MHz, 600 MHz, 800 MHz and 1000 MHz). The corresponding power consumption for each speed level is given in the second row of the table. By choosing a task with WCET equal to 100 ms with regard to the highest speed, the third row shows the respectively scaled execution time. Finally, the fourth row shows the task AEC, which is the product of the power consumption (second row) and the task run-time (third row). Clearly, the task running at 400 MHz consumes the least energy and therefore 400 MHz is the critical speed of the processor. From the practical point of view, the critical speed is in fact the most energy efficient speed concerning hertz per watt.

### 2.2.3 Interplay of DPM and DVS

The DPM and DVS techniques are widely applied in practice for run-time energy management. However, if both of them are used in a combined manner, especially in the context of system-wide energy optimization, they are actually working contradictory. The intuitive reason is that, on one hand, the DPM preferred strategy tries to complete work as soon as possible so that more idle time is available for sleeping, on the other hand, the DVS preferred strategy tries to finish task as late as possible, because lower speed is more energy efficient, at least for those above the critical speed.

Figure 2.7 compares the situations by applying the DPM preferred strategy against the DVS preferred strategy. The example CPU supports two operating frequencies $F_1$ and $F_2$, where one assumes that $F_1$ is higher and $F_2$ is the critical speed. There is only one sleep state (low power state). A real-time task with the given release time and deadline is executed on the CPU. The DPM preferred strategy selects the higher speed $F_1$ to run the task and thus the idle time is long enough to put the CPU to the sleep state. Hereby $E^1_{active}$ and $E^1_{idle}$ denote the energy consumed during the task execution and the idle time, respectively. On the contrary, the DVS preferred strategy selects the lower speed $F_2$ to execute the task, provided that the task is able to complete

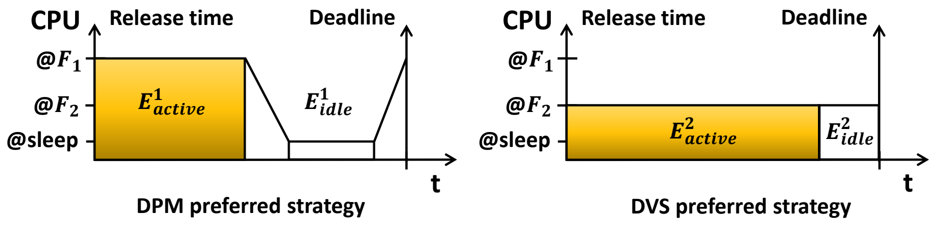


Figure 2.7: Comparison of the DPM and DVS preferred strategies

before the deadline. By definition of critical speed, $E^2_{active} < E^1_{active}$ clearly holds. However, due to the extended task execution, the idle time on the right hand side in Figure 2.7 is too short to shut down the CPU. Thus the CPU has to stay active and $E^2_{idle}$ may possibly exceed $E^1_{idle}$. If so, it is unknown whether $E^1_{active} + E^1_{idle} > E^2_{active} + E^2_{idle}$. In brief, there is a trade-off between the usage of the DPM and DVS techniques. In general, by applying the DPM technique, the possibility to use the DVS technique becomes limited and vice versa.

### 2.2.4 DPM and DVS on Multi-Core Processor Platforms

Until now, the discussion about the DPM and DVS techniques has focused on single-core processors. As multi-core processors become more and more popular, this subsection introduces the essentials of the DPM and DVS application on these platforms. In terms of the DPM and DVS capabilities, multi-core processor platforms can be divided into three categories. Their main difference is at which level the DPM and DVS techniques can be applied. They are referred to as full-chip platforms, per-core platforms and cluster-based platforms.

In the early years, full-chip platforms were commonly adopted because of the cost-efficiency of a shared power supply net. However, they lack flexibility for power management, because all the processor cores have to operate at the same voltage and speed (e.g. Intel Core™2 Quad [Intc]) at the same time. However, each individual processor core can be switched off independently (Intel Core™2 Quad [Intc]). The main reason for this is rather obvious, because switching off one processor core will not affect other processor cores and this requires only a few transistors. The voltage scaling process on the other hand is fairly expensive and needs different power lines to be pulled. In other words, for full-chip platforms the DPM technique can be applied for each processor core independently and the DVS technique can only be used at the entire chip level.

With the introduction of the Frequency/Voltage Island technique ([DNM06] [Hu+04]), per-core platforms (e.g. AMD Phenom™ Quad-Core [AMD]) gain more and more interests, where all the processor cores can be controlled

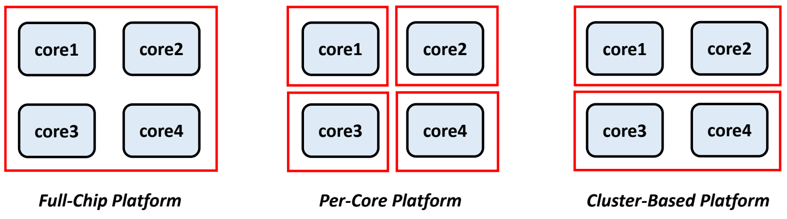


Figure 2.8: Three different categories of multi-core processor platforms

independently. However, it suffers the problem of high implementation costs, because each processor core requires a dedicated on-chip voltage regulator. This could become impractical if the number of cores dramatically increases, as in the case of many-core systems.

This dissertation focuses on the third category, the so-called cluster-based platforms, which combine the advantages of per-core and full-chip platforms and offers the best compromise. The processor cores are divided into clusters. The cores in the same cluster behave as in a full-chip platform while the cores from different clusters behave as in a per-core platform. One of the first commercial products in this context is the recently published ARM big.LITTLE™ SoCs [ARM] [Jef] composed of a cluster of Cortex-A15 cores and a cluster of Cortex-A7 cores. Clearly, the cluster-based multi-core platform is a general form of per-core and full-chip platforms. Therefore, the approach proposed for cluster-based multi-core processor platforms is applicable for the other two categories as well. Figure 2.8 illustrates the different platforms based on a quad-core processor.

## 2.3 Energy Efficient Real-Time Scheduling

Energy efficient real-time scheduling is a real-time scheduling which ensures that all the tasks will finish before their deadlines and the system energy consumption is minimized. This section shows an overview of the state-of-the-art analysis by reviewing the existing approaches for both single-core and multi-core processor platforms.

There have been extensive research work in the field of applying the DPM and DVS techniques in hard real-time systems. In the context of single-core processor platforms the existing studies can be roughly divided into three categories, DVS-only, DPM-only and DVS/DPM-combined approaches. By considering the DVS-only approaches, the primary focus is to find the optimal speed-to-task assignment to reduce processor power consumption. One of the earliest optimal offline DVS algorithms with polynomial complexity is proposed by Yao et al. [YDS95]. [ZCK07] introduced a fully polynomial time approximation algorithm. A collaborative approach based on operating system and compiler is proposed in [Abo+03], where the decision of frequency adjustment is made by means of the inserted code instruments in the application. The problem of energy-aware scheduling for non-preemptive real-time tasks is addressed in [JG05]. In order to cover the resource sharing aspect in real-time systems, the authors of [ZC02] proposed a static dual speed algorithm based on a stack resource policy. A dynamic speed adjustment scheme and a dynamic slack reclaiming algorithm are presented as well to further reduce energy consumption at run-time. [ZX06] proposed a polynomial time approximation scheme using the dynamic programming technique to statically assign frequencies to tasks while minimizing the total system-wide energy consumption. The authors of [Ayd+04] proposed a power aware real-time scheduling with three components: i) an offline algorithm to compute the optimal speed assignment, ii) an online dynamic slack reclaiming framework and iii) an online aggressive speed adjustment scheme that takes the advantage of dynamic slack of future tasks. Moreover, the work [LS04] has shown an online DVS approach by exploiting dynamic slack with a complexity of $O(1)$. A performance comparison of different DVS approaches for real-time systems under a uniform simulation framework is given in [Kim+02]. A comprehensive survey of the DVS-only approaches for single-core processor platforms is given in [CK07].

The DPM-only work is mostly proposed for device usage scheduling. The survey [BBM00] gives the basics of the DPM technique. In [CG06] an online DPM algorithm in conjunction with the EDF scheduling algorithm is proposed, where the tasks are procrastinated as much as possible to create large device idle intervals. The work by Swaminathan et al. [SC05] proposed an offline optimal device scheduling algorithm for hard real-time systems based on pruning techniques. A heuristic search algorithm is proposed as well to find a near optimal solution. The authors of [Row+08] proposed a rate-harmonized task scheduling, where an artificial task period is introduced and all the tasks are only eligible to execute at the new period boundaries. This has the effect that some ready tasks may be delayed in order to prolong the current idle time for maximizing the DPM usage. [LHC11] introduced a dynamic counter approach to decide the number of upcoming events and therefore bound the future workload. Based on this information, the devices can be safely shut sown.

Moreover, the relationship between the DPM and DVS techniques attracted more attentions in the context of system-wide energy minimization. Generally, the DVS and DPM techniques are applied for the processor and the devices, respectively. Devadas et al. [DA08b] studied the exact interplay of DPM and DVS. However, their focus is on the frame-based task model, where all the tasks share a common period/deadline. In the work of Jejurikar et al. [JG04] the concept of critical speed was applied. A task should never run below the critical speed, otherwise the power consumption increases. However, they ignored DPM state switching overhead. [DA10] introduced the idea of Device Forbidden Region (DFR), which is reserved time frame for devices staying idle. The offline part of the approach assigns all the tasks with a common speed and computes the DFR for each device with regard to system schedulability. The online part then tries to predict the length of idle time by incorporating the previously computed DFR and thus can decide whether a device can be put into a low power state. A generalized dynamic slack reclaiming technique was proposed as well to further reduce system-wide energy consumption. [Qua+04] proposed to apply the DVS technique and additionally procrastinate the ready tasks when possible, so that the processor idle time becomes longer than the break-even time. Niu has claimed in his work [Niu11] and [Niu10] that the critical speed is no longer the optimal speed, if the energy consumption during the idle time is considered and the DPM state switching overhead is non-negligible. In other words, the speeds below the critical speed could also be beneficial. The study then proposed a mechanism to adjust the task operating speed based on the so-called feasible interval, which is defined as the interval, during which a task can be executed continuously without missing its deadline. In his follow-up work [NL11], the approach is extended to take into consideration dynamic slack, which is mainly reclaimed for further usage of the DVS and DPM techniques. The authors of [Kon+10] formulated the system-wide energy optimization problem for hard real-time tasks as a 0-1 integer non-linear programming problem, however, they only focused on the frame-based task model.

In the context of multi-core processor platforms, the problem of energy efficient real-time scheduling has been discussed in several studies as well. Based on their target platforms the approaches can be roughly divided into three categories, per-core, full-chip and cluster-based approaches. Aydin et al. [AY03] showed the $\mathcal{NP}$-hardness of the problem of energy-aware task partitioning and proposed a load balancing framework based on per-core platforms. [AA05] compared various partitioning algorithms and speed assignment schemes in terms of energy consumption and feasibility analysis. Chen et al. [CK05] described a per-core DVS algorithm taking different power characteristics of tasks into consideration, however, they assumed the frame-based task model and ideal DVS model, where a processor can operate at any frequency within a given range. With regard to leakage power the same authors provided several approximation algorithms in another work [CHK06] for per-core platforms, but the analysis is based on the ideal DVS model as well. More information about their work may also be found in [Che+07] and [Che+04]. Zhu et al. [ZMC03] studied energy aware global scheduling on per-core platforms based on the dynamic slack sharing scheme. In [YP02] the authors formulated the power optimization problem on per-core platforms (each core is enabled with identical DVS capability) as an extended generalized assignment problem and tries to solve it with an extended linearization heuristic (LR-heuristic). In the same context the aspect of dependent real-time tasks is addressed in [Gru00] and [GK01] using a list-scheduling algorithm. Lee et al. [Lee09] classified the tasks into three categories according to their utilization and proposed a heuristic scheduling scheme based on this classification.

In the domain of full-chip platforms, most works are concentrating on balanced task partitioning. Seo et al. [Seo+08] introduced a dynamic repartitioning algorithm based on an existing partition, which tries to balance the task load on different cores by considering dynamic slack. A dynamic core scaling scheme was proposed as well to reduce the number of active cores. Yang et al. [YCK05] proposed an approximation algorithm to schedule a set of frame-based tasks. In the work [AB02] the authors investigated the problem of deciding the optimal number of required processors to minimize energy consumption by a given workload. However, they only considered static power management, i.e., no run-time speed adjustment is allowed and all the processors are running at a pre-computed speed.

Only recently, more and more attention has been put on cluster-based multi-core processor platforms. [KZS11] has focused on the problem of clustering processor cores into DVS domains. The authors proposed to group similar cores into clusters according to their typical workload. Chakraborty et al. [CR11] introduced a fundamentally alternative means for cluster-based multi-core processor design. They believe that a processor core, which is designed for a dedicated frequency/voltage domain, is more energy efficient than a core designed with run-time DVS capabilities and configured to that frequency/voltage domain. Both works are not focusing on hard real-time systems. Two closely related works are presented by Kong et al. [KYD11] and Qi et al. [QZ08]. The work [KYD11] addressed the energy optimization problem on cluster-based multi-core processors. Basically, their algorithm is divided into three steps: i) deciding the number of required clusters, ii) partitioning the tasks based on the worst fit strategy and iii) attempting to assign each task with the most energy efficient operating speed while meeting deadlines based on the previous partition. The authors of [QZ08] considered the same problem and even took dynamic slack into account. However, both [KYD11] and [QZ08] focused only on the frame-based task model and ideal DVS model, which are not really applicable in practice. Another closely related work is shown in [Han+12], in which shared resources with critical sections are considered. For this, a suspension based resource access protocol is developed, which means that if a task is blocked due to resource lock, it will not do a busy-wait but rather is suspended so that other tasks on the same processor core can be executed. Besides, they also proposed a task partition heuristic based on the worst fit strategy and several run-time slack management techniques including slack reclaiming, slack preserving, slack releasing and slack stealing.

As a summary, Table 2.2 and 2.3 shows the existing approaches at a glance with regard to some important aspects explained as follows.

- System-wide: This aspect shows whether an approach addresses the total system energy consumption including the processor and devices. If the main focus is only on processors or on devices, then they are explicitly given in the tables.
- DPM: This aspect indicates whether the DPM technique is applied, especially with non-negligible state switching overhead.
- DVS: This aspect gives whether an approach uses the DVS technique. If a study applies DVS but assumes an ideal DVS model, i.e., the supported operating frequencies are not discrete but rather form a continuous range, then it is explicitly given. Note that the ideal DVS model does not exist in practice.
- Online: This aspect implies whether an approach is online. If an algorithm contains an offline and an online part, it is referred to as a hybrid approach.
- Task model: This aspect shows whether the addressed task model is a general real-time task model. Some studies have applied the limited frame-based task model, where all the tasks share a common period/deadline.
- Platform: This aspect is only relevant for multi-core processors. It gives the target platform an approach has addressed. Mainly there are three types of them: per-core, full-chip and cluster-based platforms.

These aspects also highlight the main contribution of the dissertation. The main distinction between this dissertation and the existing work is that this work addressed all the aspects, i.e., system-wide, DPM with non-negligible switching overhead, non-ideal DVS model, online, general real-time task model and cluster-based multi-core processor platforms.

## 2.4 Advanced Configuration & Power Interface

Advanced Configuration & Power Interface (ACPI) is an industrial open standard for unifying hardware/software interaction interfaces [ACP]. The primary objective is to enable operating system directed device configuration and power management. Figure 2.9 shows the overall structure of an ACPI compatible system. In particular, an ACPI subsystem is composed of three major parts: ACPI register, ACPI BIOS and ACPI tables [ACP]. The last one plays a key role to connect software and hardware components.

- **ACPI registers** are special hardware registers for ACPI subsystem to configure a system.

| Related work | System-wide | DPM | DVS | Online | Task model |
|---|---|---|---|---|---|
| [YDS95] | processors | x | Ideal | x | ✓ |
| [ZCK07] | processors | x | ✓ | x | ✓ |
| [Abo+03] | processors | x | ✓ | Hybrid | ✓ |
| [JG05] | processors | x | ✓ | Hybrid | ✓ |
| [ZC02] | processors | x | ✓ | Hybrid | ✓ |
| [ZX06] | ✓ | x | ✓ | x | ✓ |
| [Ayd+04] | processors | x | Ideal | Hybrid | ✓ |
| [LS04] | processors | x | Ideal | ✓ | ✓ |
| [CG06] | devices | ✓ | x | ✓ | ✓ |
| [SC05] | devices | ✓ | x | x | ✓ |
| [Row+08] | devices | ✓ | x | x | ✓ |
| [LHC11] | processors | ✓ | x | ✓ | ✓ |
| [DA08b] | ✓ | ✓ | Ideal | x | Frame-based |
| [JG04] | ✓ | x | ✓ | x | ✓ |
| [DA10] | ✓ | ✓ | ✓ | Hybrid | ✓ |
| [Qua+04] | processors | ✓ | ✓ | Hybrid | ✓ |
| [Niu11] | ✓ | ✓ | ✓ | Hybrid | ✓ |
| [Niu10] | processors | ✓ | ✓ | Hybrid | ✓ |
| [NL11] | processors | x | ✓ | Hybrid | ✓ |
| [Kon+10] | ✓ | ✓ | ✓ | x | Frame-based |

Table 2.2: A comparison among the existing approaches (single-core processor)

| Related work | Platform | DPM | DVS | Online | Task model |
|---|---|---|---|---|---|
| [AY03] | Per-core | x | Ideal | x | ✓ |
| [AA05] | Per-core | x | Ideal | x | ✓ |
| [CK05] | Per-core | x | Ideal | x | Frame-based |
| [CHK06] | Per-core | ✓ | Ideal | x | ✓ |
| [ZMC03] | Per-core | x | Ideal | ✓ | ✓ |
| [YP02] | Per-core | x | ✓ | x | ✓ |
| [Gru00] | Per-core | x | ✓ | x | ✓ |
| [Che+04] | Per-core | x | Ideal | x | Frame-based |
| [Lee09] | Per-core | x | ✓ | x | ✓ |
| [Seo+08] | Full-chip | x | Ideal | ✓ | ✓ |
| [YCK05] | Full-chip | x | Ideal | x | Frame-based |
| [AB02] | Full-chip | x | ✓ | Hybrid | ✓ |
| [KYD11] | Cluster-based | x | Ideal | x | Frame-based |
| [QZ08] | Cluster-based | x | Ideal | Hybrid | Frame-based |

Table 2.3: A comparison among the existing approaches (multi-core processor)

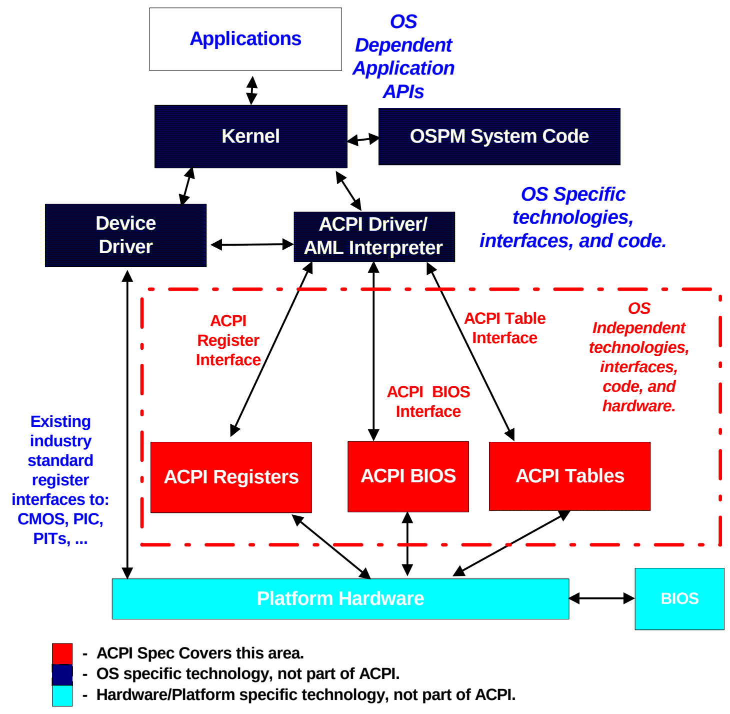

Applications
OS Dependent Application APIs
Kernel
OSPM System Code
Device Driver
ACPI Driver/ AML Interpreter
OS Specific technologies, interfaces, and code.
ACPI Register Interface
ACPI Table Interface
OS Independent technologies, interfaces, code, and hardware.
ACPI BIOS Interface
Existing industry standard register interfaces to: CMOS, PIC, PITs, ...
ACPI Registers
ACPI BIOS
ACPI Tables
Platform Hardware
BIOS
🟥 - ACPI Spec Covers this area.
🟦 - OS specific technology, not part of ACPI.
🟦 - Hardware/Platform specific technology, not part of ACPI.

Figure 2.9: ACPI System Structure [ACP]

- **ACPI BIOS** describes the portion of BIOS that is compatible with ACPI standard. It provides, for instance, interfaces to an operating system for sleep, wake-up and restart operations.

- **ACPI tables** is the main part of an ACPI subsystem and contains the descriptions of hardware components that are used by an operating system. Hereby the ACPI specification only defines the interface of functionality and the implementation is often left open to vendors. There are different types of ACPI tables with different purposes. The basic element in a table is the so-called definition block. ACPI tables can be programmed by using the ACPI Source Language (ASL) and later compiled to the ACPI Machine Language (AML), a pseudo-code type of language. Thus an operating system needs an interpreter to understand and execute the functionality specified in the tables. One of the most important tables is the Differentiated System Description Table (DSDT) containing the most hardware information.

The ACPI specification is a fairly large standard. This dissertation focuses on a small part of it, namely the power management. In particular, the ACPI standard defines the power management functionality for processors and devices.

For processor power management, power and performance states are defined in the ACPI specification [ACP]. The processor power states are usually referred to as C-states and used to describe different operation modes. They are designated by $C_0, C_1, C_2, ..., C_n$. $C_0$ is a mandatory state and also the only state, in which a processor may execute software programs. $C_1$ is specified as a mandatory state as well and should be supported by a special instruction of a processor architecture, such as `HLT` for the IA-32 architecture. In $C_1$, the content of caches should be preserved. Furthermore, $C_2$ is an optional state which should provide more power saving than $C_1$. In $C_2$, a processor should be able to keep its caches coherent. The $C_3$ state is an optional state as well. If it is present, it should provide even more power saving than $C_2$ and have longer entry and exit latency. In $C_3$ the context in a processor cache is no longer maintained. Further C-states can also be supported by a processor. It is then left open to vendors. If they are present, their information is available in the ACPI tables and can be read by an operating system. The concept of C-states is comparable to the DPM states. Besides the processor power states, the ACPI standard further defines processor performance states as well, which are commonly referred to as P-states. Hereby the main idea is similar to the DVS states, which describe different performance levels of a processor.

For device power management, ACPI defines D-states to capture different power states of a device. This is similar to the processor power states, where there is only one active state. The remaining states are used then for energy saving when the device is idle. It follows the same principle, the more energy saving, the longer it takes to enter and exit the state. In fact, D-states represent the DPM states for devices.

## 2.5 Chapter Summary

This chapter gave the background of the dissertation. First, a short introduction of real-time systems and scheduling is shown. Second, two run-time energy management techniques, DPM and DVS, are thoroughly reviewed. In particular, the main problem when using DPM and DVS together in the context of hard real-time systems is discussed. In addition, the three types of multi-core processor platforms in terms of the DPM and DVS capabilities are explained. After that, a state-of-the-art analysis regarding energy efficient real-time scheduling is presented. Finally, before this chapter is concluded, a brief overview of the ACPI standard is given.

# Chapter 3

# System Models and Problem Formulation

This chapter formally introduces basic notions and terminologies and defines the fundamental system models with special focus on power aspects. In the context of this dissertation, the hardware of a system is composed of a processor and multiple I/O devices, such as hard disk, ethernet controller, flash card controller and other possible peripherals. The processor power model and device power model are described in the first and second section, respectively. This dissertation assumes that a processor supports both DPM and DVS techniques while a device is only equipped with DPM capability. This assumption is widely applied in the context of system-wide energy optimization and can be found in numerous other studies, e.g., [DA10], [NL11] and [CG09]. Moreover, a real-time task model is described as well to address the software components of a system. Finally, before this chapter is concluded, the energy optimization problems in hard real-time systems are formulated.

## 3.1 Processor Power Model

Since this work primarily targets energy optimization, the definitions here concentrate on the power characteristics of processors rather than the functional aspects. The processor power model is defined following the ACPI recommendation by adopting the concept of C-states and P-states explained in the previous chapter. The C-states mainly describe different power states containing one active state and multiple low power (sleep) states with different sleep depth while the P-states reveal different performance states with the corresponding operating speed. In other words, C-states represent states related to the DPM technique and P-states can be identified as states with regard to DVS. Therefore, in the remaining text the C-states and P-states are also referred to as the DPM states and DVS states, respectively. In what follows, the power model of single-core processors and multi-core processors are defined in detail.

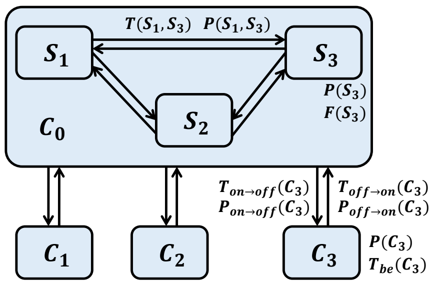


Figure 3.1: The power state machine of a processor with 3 P-states and 4 C-states

### 3.1.1 Single-Core Processor

Let $\mathcal{C} = \{C_0, C_1, ..., C_c\}$ denote a finite set of C-states of a processor, where $C_0$ is the only working state, i.e., the processor can only execute tasks in this state and $C_1, C_2, ..., C_c$ are low power states (sleep states) in non-increasing order of their power consumption. Moreover, the state $C_0$ is defined as a superstate that contains a finite set of P-states denoted by $\mathcal{S} = \{S_1, S_2, ..., S_s\}$. $S_1$ is the full performance state with the maximal operating speed and the remaining states are sorted in non-increasing order of their power consumption. $\forall i$ : $1 \leq i \leq s$, $F(S_i)$ and $P(S_i)$ denote the corresponding frequency (normalized with regard to $F(S_1)$) and the power consumption, respectively. For instance, if a processor supports two operating speeds 100 MHz and 50 MHz, then $F(S_1) = 1$ and $F(S_2) = 0.5$. The overhead for state switching among different P-states is defined as follows:

- $P(S_i, S_j)$ is the power consumption of state switching from $S_i$ to $S_j$.
- $T(S_i, S_j)$ is the latency of state switching from $S_i$ to $S_j$.

Furthermore, $\forall i : 1 \leq i \leq c$, $P(C_i)$, $T_{on\to off}(C_i)$, $T_{off\to on}(C_i)$, $P_{on\to off}(C_i)$, $P_{off\to on}(C_i)$ and $T_{be}(C_i)$ are defined as follows:

- $P(C_i)$ is the power consumption of the state $C_i$.
- $T_{on\to off}(C_i)$ is the latency for state switching from $C_0$ to $C_i$.
- $T_{off\to on}(C_i)$ is the latency for state switching from $C_i$ to $C_0$.
- $P_{on\to off}(C_i)$ is the power consumption for state switching from $C_0$ to $C_i$.
- $P_{off\to on}(C_i)$ is the power consumption for state switching from $C_i$ to $C_0$.

- $T_{be}(C_i)$ is the break-even time to enter the state $C_i$.

In general, the C- and P-states of a processor can be modeled as a so-called power state machine [BBM00]. Figure 3.1 shows an example power state machine of a processor with 3 P-states and 4 C-states. In the figure the power properties of the state $C_3$ are illustrated. As observed, this dissertation assumes that C-state switching may only take place between the active state and the low power states. A switching between two low power states is not allowed. In addition, if a processor needs to be switched to a low power state, then the current P-state is remembered as the history state that will be entered when the processor wakes up later.

The definition of break-even time is introduced in the previous chapter and its calculation formula is given in (2.20). Thus, $\forall i : 1 \leq i \leq c, T_{be}(C_i)$ can be derived as follows.

$$T_{be}(C_i) = \max\{T_{overhead}(C_i), \frac{E_{overhead}(C_i) - P(C_i) \cdot T_{overhead}(C_i)}{P_{on} - P(C_i)}\} \tag{3.1}$$

where $T_{overhead}(C_i)$ and $E_{overhead}(C_i)$ are defined in (3.2) and (3.3), respectively. Intuitively, they are the latency and energy consumption required to enter and exit the low power state.

$$T_{overhead}(C_i) = T_{off \to on}(C_i) + T_{on \to off}(C_i) \tag{3.2}$$

$$E_{overhead}(C_i) = T_{off \to on}(C_i) \cdot P_{off \to on}(C_i) + T_{on \to off}(C_i) \cdot P_{on \to off}(C_i) \tag{3.3}$$

Furthermore, $P_{on}$ denotes the power consumption when the processor is in the active state $C_0$. As $C_0$ is a superstate, the value of $P_{on}$ is clearly dependent on the concrete P-state. In other words, $T_{be}(C_i)$ becomes different if different P-states are selected. As break-even time is defined as the minimum time required to stay in a low power state, choosing the maximal value of $T_{be}(C_i)$ keeps the computation on the safe side. Thus, the final calculation formula of $T_{be}(C_i)$ is derived.

$$T_{be}(C_i) = \max_{S_j \in \mathcal{S}}\{T_{overhead}(C_i), \frac{E_{overhead}(C_i) - P(C_i) \cdot T_{overhead}(C_i)}{P(S_j) - P(C_i)}\} \tag{3.4}$$

### 3.1.2 Multi-Core Processor

The multi-core processor power model is a generalization of the single-core processor power model, where a finite set of processor cores are denoted by $\mathcal{O} = \{O_1, O_2, ..., O_o\}$. Each processor core $O_i$ is represented by a pair $O_i = (\mathcal{C}^i, \mathcal{S}^i)$, where $\mathcal{C}^i$ and $\mathcal{S}^i$ denote the set of its C-states and P-states, respectively. More concretely, $\mathcal{C}^i = \{C_0^i, C_1^i, ..., C_{c_i}^i\}$ and $\mathcal{S}^i = \{S_1^i, S_2^i, ..., S_{s_i}^i\}$ hold. The power consumption and switching overhead of each state are defined in the same way as in the single-core processor power model.

Furthermore, this dissertation considers cluster-based multi-core processors, where processor cores are clustered into disjoint groups $\mathcal{G} = \{G_1, G_2, ..., G_g\}$ with

$$\mathcal{O} = \bigcup_{i=1}^{g} G_i \tag{3.5}$$

$$\forall G_i, G_j : i \neq j \Rightarrow G_i \cap G_j = \varnothing \tag{3.6}$$

and

$$\forall G_i : G_i \in \mathcal{G}, G_i \subseteq \mathcal{O} \tag{3.7}$$

To formally define the clustering, the function $group : \mathcal{O} \rightarrow \mathcal{G}$ is introduced. Hereby $group(O_i)$ expresses the group containing $O_i$. One important constraint on cluster-based multi-core processor is that the cores in the same cluster may only operate at the same speed. In case of a speed conflict, i.e., two cores in one cluster require two different speeds, the highest speed is used as the cluster wide operating speed. This speed coordination strategy, on the one hand, is applied due to hard real-time constraints (more details are shown later) and on the other hand is even obliged on some platforms due to hardware restriction, e.g., Intel Core™ 2 Quad [Intc]. Instinctively, this work assumes that the processor cores in the same cluster share a common power model, i.e., $\forall O_i \in \mathcal{O}, O_j \in \mathcal{O}$, if $group(O_i) = group(O_j)$, then $O_i = O_j$.

## 3.2 Device Power Model

A finite set of devices is denoted by $\mathcal{R} = \{R_1, R_2, ..., R_m\}$ and their power models are defined following the D-states concept in the ACPI specification. More specifically, $\mathcal{D}^i = \{D_0^i, D_1^i, ..., D_{d_i}^i\}$ denotes the finite set of power states of the device $R_i$. $D_0^i$ is the only active state and the remaining states are low

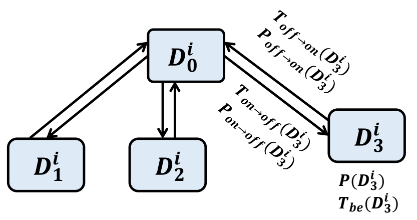


Figure 3.2: The power state machine of $R_i$ with 4 D-states

power states (sleep states) in non-increasing order of the power consumption. In fact, the D-states are the DPM states for devices. Similar to the processor power model, state switching is only allowed between the active state and the low power states. Figure 3.2 shows a simple power state machine of a device $R_i$ with 4 D-states. Furthermore, the power consumption of each state is denoted by $P(D_j^i)$. For each low power state $D_j^i$ with $j > 0$, $T_{on\to off}(D_j^i)$, $T_{off\to on}(D_j^i)$, $P_{on\to off}(D_j^i)$, $P_{off\to on}(D_j^i)$ and $T_{be}(D_j^i)$ are defined as follows.

- $T_{on\to off}(D_j^i)$ is the latency for state switching from $D_0^i$ to $D_j^i$.
- $T_{off\to on}(D_j^i)$ is the latency for state switching from $D_j^i$ to $D_0^i$.
- $P_{on\to off}(D_j^i)$ is the power consumption for state switching from $D_0^i$ to $D_j^i$.
- $P_{off\to on}(D_j^i)$ is the power consumption for state switching from $D_j^i$ to $D_0^i$.
- $T_{be}(D_j^i)$ is the break-even time to enter the state $D_j^i$.

In Figure 3.2, the power properties of the state $D_3^i$ are illustrated in detail. Due to the assumption that the DVS technique is not supported by devices, the active state here is a normal state, as opposed to the superstate in the processor power model. Thus the break-even time of each low power state can be computed more easily. By applying (2.20), $\forall i, j$ with $j > 0$, the formula of $T_{be}(D_j^i)$ is shown in (3.8).

$$T_{be}(D_j^i) = \max\{T_{overhead}(D_j^i), \frac{E_{overhead}(D_j^i) - P(D_j^i)\cdot T_{overhead}(D_j^i)}{P(D_0^i) - P(D_j^i)}\} \quad (3.8)$$

where $T_{overhead}(D_j^i)$ and $E_{overhead}(D_j^i)$ are defined in (3.9) and (3.10), respectively. Intuitively, they present the latency and energy consumption required to enter and exit the low power state.

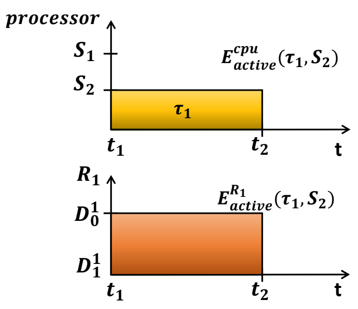


Figure 3.3: An example task execution at the speed level $S_2$ with $Dev(\tau_1) = \{R_1\}$

$$T_{overhead}(D^i_j) = T_{off\rightarrow on}(D^i_j) + T_{on\rightarrow off}(D^i_j) \tag{3.9}$$

$$E_{overhead}(D^i_j) = T_{off\rightarrow on}(D^i_j) \cdot P_{off\rightarrow on}(D^i_j) + T_{on\rightarrow off}(D^i_j) \cdot P_{on\rightarrow off}(D^i_j) \tag{3.10}$$

## 3.3 Real-Time Task Model

In this work the traditional real-time task model from [But11] is adopted. A set of independent tasks is denoted by $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$ with $W(\tau_i)$ denoting the WCET at the maximal (without loss of generality) processor speed $S_1$ (i.e., the actual WCET of a task $\tau_i$ running at another speed $S_j$ is $\frac{W(\tau_i)\cdot F(S_1)}{F(S_j)}$), $T(\tau_i)$ denoting the relative deadline (equal to the period) and $Dev(\tau_i)$ denoting the set of devices required by the task execution, i.e., $Dev(\tau_i) \subseteq \mathcal{R}$. $\forall R_j \in Dev(\tau_i)$, this dissertation assumes that $R_j$ must stay active during the entire execution time of the task $\tau_i$. Figure 3.3 shows the task execution of $\tau_1$ in the time $[t_1, t_2]$. The example further assumes $Dev(\tau_1) = \{R_1\}$. Thus $R_1$ stays active during $[t_1, t_2]$. Note that this work assumes non-existence of critical sections by device access. However, Chapter 9 will discuss possible extensions of this work to address this issue. The hyper period of a task set is denoted by $HP$, which is the least common multiple of all the task periods.

As shown in the previous chapter by (2.23), AEC of a task is a convex function of speed and the concept of critical speed is introduced. No task should ever run below that speed, because otherwise both the AEC and execution time of the task increase. In this section, the definition of the task AEC is extended by incorporating the energy consumption of devices.

**Definition 3.3.1.** *The **Active Energy Consumption (AEC)** of a periodic task $\tau_i$, denoted by $E_{active}(\tau_i, S)$, is defined as the energy consumed by the pro-*

*cessor and all the required devices during the task execution at a particular speed state S within one period.*

An example is illustrated in Figure 3.3, where $\tau_1$ requires $R_1$ during its execution. It is assumed that $\tau_1$ runs at the speed state $S_2$. Thus, the AEC of $\tau_1$ is composed of two parts, the energy consumed by the processor $E^{cpu}_{active}(\tau_1, S_2)$ and the energy consumption of the device $E^{R_1}_{active}(\tau_1, S_2)$. To generalize the description, the AEC of a task $\tau$ running at a particular speed state $S_i$ can be computed as follows.

$$\begin{aligned} E_{active}(\tau, S_i) &= E^{cpu}_{active}(\tau, S_i) + \sum_{R_j \in Dev(\tau)} E^{R_j}_{active}(\tau, S_i) \\ &= P(S_i) \cdot \frac{W(\tau) \cdot F(S_1)}{F(S_i)} + \sum_{R_j \in Dev(\tau)} P(D^j_0) \cdot \frac{W(\tau) \cdot F(S_1)}{F(S_i)} \end{aligned} \quad (3.11)$$

Obviously, if $Dev(\tau) = \varnothing$, the task AEC here becomes the same as defined in Definition 2.2.1 in Chapter 2. Within this context, the critical speed for each task $\tau$, denoted by $SC(\tau)$, can be calculated as the most power efficient P-state in terms of its AEC, i.e.,

$$\forall S_i \in \mathcal{S} : E_{active}(SC(\tau), \tau) \leq E_{active}(S_i, \tau) \quad (3.12)$$

Note that, since device energy consumption is taken into consideration and different tasks may require different sets of devices, a critical speed becomes task-dependent, i.e., each task may have a different critical speed.

In the context of multi-core processor platforms, if the processor cores have different power models, then the critical speed for each task is dependent on the task partition, i.e., on which processor core the task is executed. Therefore, for a particular task $\tau$, $SC^j(\tau)$ denotes its critical speed on the processor core $O_j$, i.e.,

$$\forall S^j_i \in \mathcal{S}^j : E_{active}(SC^j(\tau), \tau) \leq E_{active}(S^j_i, \tau) \quad (3.13)$$

As a short summary, the critical speed is task-dependent, because each task may require different sets of devices. In a multi-core processor, it is also core-dependent, provided that the processor cores have heterogeneous power models. By definition, the critical speed of a task is the optimal speed in terms of its AEC. If a task runs at the critical speed, it is a good indicator that the task execution is energy efficient. However, a system may contain many tasks and all the tasks running at their critical speeds do not necessarily yield the minimal system energy consumption. The main reason is, on the one hand, due to the energy consumption of idle time, which is not addressed in the concept of critical speed. On the other hand, the calculation of critical speed disregards schedulability issue, i.e., running all the tasks at the respective critical speeds may result in a deadline miss. As will be seen later in Chapter 4 and 5, the task critical speed is often applied only as heuristic information to find the actually optimal operating speed for tasks.

## 3.4 Problem Formulation

In general, the main objective is to find the most appropriate way to apply the DPM and DVS techniques to save system-wide energy consumption while meeting all the task deadlines. As indicated in the previous chapter, both DPM and DVS reduce energy at the cost of performance degradation. Thus they have to be used carefully. Roughly speaking, the key challenge of the DPM application is to predict the length of idle time and select the most proper low power state, whereas the challenge of the DVS application is to assign each task with the most proper operating speed, so that the system energy consumption is minimized. Note that the trade-off between the DPM and DVS usages is to be considered as well. Before formally formulating the problems, several definitions are first introduced.

**Definition 3.4.1.** *Given a task set $\Gamma$ and a set of available P-states $\mathcal{S}$ on a processor core, a* ***speed assignment*** *is defined as a function from the task set to the P-states. It is denoted by* $\boldsymbol{assign}: \boldsymbol{\Gamma} \rightarrow \boldsymbol{\mathcal{S}}$.

This work assumes that the task execution time is linear to processor speed. For a given speed assignment, the WCET of a task $\tau_i$ is scaled up according to the assigned processor frequency. It can be easily computed by $\frac{W(\tau_i)\cdot F(S_1)}{F(assign(\tau_i))}$.

**Definition 3.4.2.** *Given a task set $\Gamma$ and a multi-core processor model $\mathcal{O}$, a* ***task allocation*** *is defined as a function which partitions the tasks onto the processor cores. It is denoted by* $\boldsymbol{alloc}: \boldsymbol{\Gamma} \rightarrow \boldsymbol{\mathcal{O}}$.

Since this work concentrates on independent tasks, the communication overhead among processor cores is ignored.

**Definition 3.4.3.** *A* ***power state scheduling*** *is defined as a scheduling, which decides at run-time when a component can be switched off and on.*

As shown in the previous chapter, shutting down a component is not only dependent on when it is idle, but also on how long the upcoming idle time takes. Only if the idle time is longer than the corresponding break-even time, the component is able to enter a low power state. Besides, due to non-negligible wake-up latency, the component needs to be switched on a little ahead of the actually required time. A power state scheduling exactly takes care of this problem.

Now the problem can be formulated for single-core processor and multi-core processor platforms, respectively.

### 3.4.1 Problem for Single-Core Processor Platforms (RTSC)

In this context, the system-wide energy consumption is of utmost importance, i.e., the energy consumed by the processor and peripheral devices must be optimized as a whole. To simplify the explanation, the term ”component” is used in a general sense and can be referred to either as a processor or a device. Roughly, the energy optimization problem for hard real-time systems on single-core processor platforms is mainly composed of two parts: the speed assignment and the task scheduling. More concretely, the first part decides the execution speed of each task while the second part determines the execution order. To find the optimal solution in general, the two parts have to be considered coherently, because they have influence on each other. Since the second part is a well-studied area and there exist many sophisticated algorithms like EDF and RM, this work mainly deals with the first part. In summary, the energy optimization problem for hard real-time systems on single-core processor platforms (RTSC) is defined as follows.

**Problem 3.4.1** (**RTSC**). *Given a single-core processor based system model, denoted by a tuple* $(\Gamma, \mathcal{C}, \mathcal{S}, \mathcal{R})$*, and a real-time scheduling algorithm, the goal is to find the optimal speed assignment* ***assign****, which ensures system schedulability and the energy consumption over a hyper period is minimal. A power state scheduling for all the components needs to be derived as well.*

Intuitively, $assign(\tau)$ gives the assigned speed state, in which the task $\tau$ is executed. In other words, the operating frequency for $\tau$ is $F(assign(\tau))$. Therefore, the actual WCET of each task $\tau$ can be expressed by $\frac{W(\tau)\cdot F(S_1)}{F(assign(\tau))}$. Note that $W(\tau)$ is defined in the previous section as the WCET with regard to $F(S_1)$. Based on this information, system schedulability can be tested for the well-known real-time scheduling algorithm EDF or RM by checking whether the total system utilization is under the given bound.

$$\sum_{\tau \in \Gamma} \frac{W(\tau)\cdot F(S_1)}{F(assign(\tau))\cdot T(\tau)} < U_B \tag{3.14}$$

For the EDF and RM real-time scheduling algorithms, the utilization bound $U_B$ is 1 and 0.69, respectively [But11]. In brief, the main challenge of RTSC

| State | $F$ | $P$ (mW) | $T_{on\to off}$ | $T_{off\to on}$ | $P_{on\to off}$ | $P_{off\to on}$ | $T_{be}$ |
|---|---|---|---|---|---|---|---|
| $S_1$ | 1 | 800 | n.a.[1] | n.a. | n.a. | n.a. | n.a. |
| $S_2$ | 0.5 | 300 | n.a. | n.a. | n.a. | n.a. | n.a. |
| $C_1$ | n.a. | 50 | 2 ms | 2 ms | 50 mW | 50 mW | 4 ms |
| $D_0^1$ | n.a. | 1000 | n.a. | n.a. | n.a. | n.a. | n.a. |
| $D_1^1$ | n.a. | 100 | 2 ms | 2 ms | 100 mW | 100 mW | 4 ms |

Table 3.1: The processor and device power model

| Task | $W$ (ms) | $T$ (ms) | $Dev$ | $SC$ |
|---|---|---|---|---|
| $\tau_1$ | 5 | 20 | $\varnothing$ | $S_2$ |
| $\tau_2$ | 10 | 40 | $\{R_1\}$ | $S_1$ |

Table 3.2: The task specification

is to find an optimal speed assignment to achieve minimal system-wide energy consumption while meeting all the hard real-time constraints.

Unfortunately, this problem has been proven to be $\mathcal{NP}$-hard in the strong sense. In the work [DA10], the authors have shown that the energy optimization problem even in the absence of the DVS application is already $\mathcal{NP}$-hard in the strong sense. The basic idea is to first define the corresponding decision variant of the optimization problem and then prove it to be $\mathcal{NP}$-hard in the strong sense by making a reduction from the 3-partition problem. As a result, both a pseudo-polynomial algorithm and a fully polynomial time approximation scheme, which are the best algorithms one can hope for a $\mathcal{NP}$-hard problem, are not existing, unless $\mathcal{NP} = \mathcal{P}$ [GJ90].

Table 3.1 and 3.2 show the system model of a simple example, which is composed of a single-core processor (with 2 DVS states and 1 low power state), a device and two real-time tasks.

By assuming that the EDF algorithm is applied in this example, the optimal solution can be easily calculated, which assigns $S_2$ to the task $\tau_1$ and $S_1$ to the task $\tau_2$. Clearly, all the tasks can be successfully scheduled as shown in Figure 3.4 (processor utilization is 0.75) and the system-wide energy consumption $E_{sys}$ over a hyper period ($HP = 40$ ms) is minimal.

$$E_{sys} = E_p + E_{R_1} \tag{3.15}$$

where $E_p$ and $E_{R_1}$ denote the energy consumed by the processor and $R_1$, respectively.

[1]not applicable

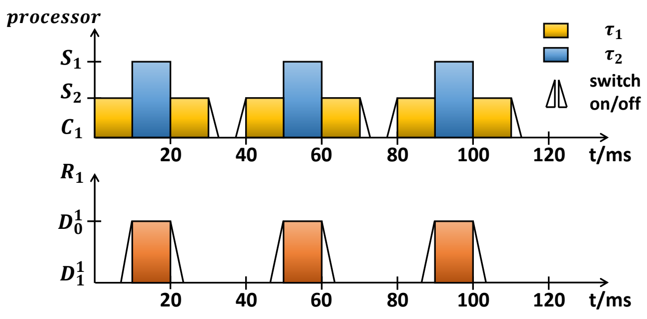


Figure 3.4: The task execution on a single-core processor

$$\begin{aligned} E_p &= 2 \cdot 10 \text{ ms} \cdot 300 \text{ mW} + 10 \text{ ms} \cdot 800 \text{ mW} \\ &\quad + 2 \cdot 2 \text{ ms} \cdot 50 \text{ mW} + 6 \text{ ms} \cdot 50 \text{ mW} \\ &= 14.5 \text{ mJ} \end{aligned} \tag{3.16}$$

and

$$\begin{aligned} E_{R_1} &= 10 \text{ ms} \cdot 1000 \text{ mW} + 2 \cdot 2 \text{ ms} \cdot 100 \text{ mW} + 26 \text{ ms} \cdot 100 \text{ mW} \\ &= 13 \text{ mJ} \end{aligned} \tag{3.17}$$

### 3.4.2 Problem for Multi-Core Processor Platforms (RTMC)

The actual energy optimization problem on a multi-core processor is mainly composed of three parts: the task partitioning (processor affinity), the frequency assignment to tasks and the task scheduling on each processor core. Note that the partitioned scheduling is considered in this context. Similar to the RTSC problem, hereby the dissertation concentrates on the first two parts and assume that the last part is solved by a given real-time scheduling algorithm like EDF or RM. Thus, the problem is stated as follows.

**Problem 3.4.2 (RTMC).** *Given a multi-core processor based system model, denoted by a tuple $(\Gamma, \mathcal{O})$ and a real-time scheduling algorithm on each core, the output is to find an optimal solution including a task partition **alloc** and a speed assignment **assign**, so that all tasks are schedulable and the system energy consumption over one hyper period is minimal. A power state scheduling needs to be derived as well.*

To simplify the problem, in this case the main focus is on the processor energy optimization and the devices are not considered, i.e., $\forall \tau : Dev(\tau) = \varnothing$.

| State | $F$ | $P$ (mW) | $T_{on\to off}$ | $T_{off\to on}$ | $P_{on\to off}$ | $P_{off\to on}$ | $T_{be}$ |
|---|---|---|---|---|---|---|---|
| $S_1^1$ | 1 | 800 | n.a. | n.a. | n.a. | n.a. | n.a. |
| $S_2^1$ | 0.5 | 300 | n.a. | n.a. | n.a. | n.a. | n.a. |
| $C_1^1$ | n.a. | 50 | 2 ms | 2 ms | 50 mW | 50 mW | 4 ms |
| $S_1^2$ | 1 | 800 | n.a. | n.a. | n.a. | n.a. | n.a. |
| $S_2^2$ | 0.5 | 300 | n.a. | n.a. | n.a. | n.a. | n.a. |
| $C_1^2$ | n.a. | 50 | 2 ms | 2 ms | 50 mW | 50 mW | 4 ms |

Table 3.3: The power model of a dual-core processor

| Task | $W$ (ms) | $T$ (ms) | $Dev$ | $SC^1$ | $SC^2$ |
|---|---|---|---|---|---|
| $\tau_1$ | 10 | 20 | $\varnothing$ | $S_2^1$ | $S_2^2$ |
| $\tau_2$ | 15 | 40 | $\varnothing$ | $S_2^1$ | $S_2^2$ |

Table 3.4: The task specification

The main reason is that shared resource access in multi-core environment is very complex and out of the scope of the dissertation. However, the conclusion chapter will give a short discussion how it could be taken into consideration.

Similar to the previous RTSC problem, hereby system schedulability can be checked by using the utilization test.

$$\forall O_i \in \mathcal{O} : \sum_{alloc(\tau)=O_i} \frac{W(\tau)\cdot F(S_1^i)}{F(assign(\tau))\cdot T(\tau)} < U_B^i \tag{3.18}$$

where $U_B^i$ is the utilization upper bound of the given real-time scheduling algorithm on the processor core $O_i$. At this point, the previously mentioned speed coordination comes into play. That is, the processor cores in the same cluster may only operate at the same speed. If a speed conflict occurs, the highest one is always selected. In this way, a task may only run at the speed higher than its assigned speed and thus no task will complete later than its original WCET, which is an important precondition for the test in (3.18). Moreover, the RTMC problem is obviously a generalized version of the RTSC problem and therefore it is NP-hard in the strong sense as well.

To simplify the explanation, a simple example with a dual-core processor and two real-time tasks is shown in Table 3.3 and 3.4, respectively. Hereby both cores $O_1$ and $O_2$ are in the same cluster and thus they have the same power model, i.e., $group(O_1) = group(O_2)$. Since in multi-core scenarios no peripheral device is considered, the set of required devices for all the tasks is empty.

By assuming that the optimal solution of this example is $alloc(\tau_1) = O_1$,

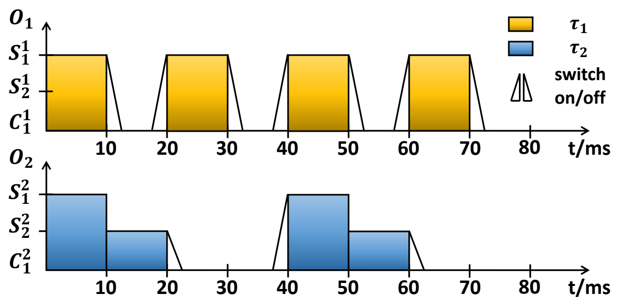


Figure 3.5: The task execution of the multi-core processor example

$alloc(\tau_2) = O_2$, $assign(\tau_1) = S_1^1$ and $assign(\tau_2) = S_2^2$ and the EDF real-time scheduling algorithm is applied on both processor cores, the task execution is shown in Figure 3.5. Clearly, all the tasks are feasible, since the utilization of both processor cores are less than 1.

$$U(O_1) = \frac{W(\tau_1) \cdot F(S_1^1)}{F(S_1^1) \cdot T(\tau_1)} = \frac{10 \text{ ms} \cdot 1}{1 \cdot 20 \text{ ms}} = 0.5 < 1 \tag{3.19}$$

$$U(O_2) = \frac{W(\tau_2) \cdot F(S_1^2)}{F(S_2^2) \cdot T(\tau_2)} = \frac{15 \text{ ms} \cdot 1}{0.5 \cdot 40 \text{ ms}} = 0.75 < 1 \tag{3.20}$$

Hereby an important observation is that the task $\tau_2$ is running at $S_1^2$ during the time interval $[0 \text{ ms}, 10 \text{ ms}]$, even it is assigned with $S_2^2$. This is due to the hardware constraints that both cores are in the same cluster and may only operate at the same speed. In other words, $\tau_2$ is enforced to run at the higher speed state $S_1^2$.

The processor energy consumption over a hyper period ($HP = 40$ ms) can be calculated as follows.

$$E = E_{O_1} + E_{O_2} \tag{3.21}$$

where

$$\begin{aligned} E_{O_1} &= 2 \cdot 10 \text{ ms} \cdot 800 \text{ mW} + 4 \cdot 2 \text{ ms} \cdot 50 \text{ mW} + 12 \text{ ms} \cdot 50 \text{ mW} \\ &= 17 \text{ mJ} \end{aligned} \tag{3.22}$$

and

$$\begin{aligned} E_{O_2} &= 10\text{ ms}\cdot 800\text{ mW} + 10\text{ ms}\cdot 300\text{ mW} \\ &\quad + 2\cdot 2\text{ ms}\cdot 50\text{ mW} + 16\text{ ms}\cdot 50\text{ mW} \\ &= 12\text{ mJ} \end{aligned} \tag{3.23}$$

## 3.5 Chapter Summary

This chapter formally defined the system models and basic terminologies. In particular, the power model of processors and devices are presented following the ACPI standard. The model for hard real-time tasks is adopted from [But11]. Finally, before this chapter is concluded, the main problems RTSC and RTMC, which are to be addressed in this dissertation, are formulated.

The system models presented in this chapter are rather formal. In order to interpret them in a more practical way, some UML artifacts could be applied. For instance, the power state machine of a processor or device can be expressed by a UML statechart. Since this topic is out of scope of this dissertation, it will not be further discussed. However, some general information and ideas can be found in [HMM11], [MHM10b], [Van+11], [MHM11], [Mul+10] and [MHM10a].

# Chapter 4

# Guided Search Algorithm based on Simulated Annealing

This chapter presents the main algorithm to solve the RTSC and RTMC problems introduced in the previous chapter. Unfortunately, both problems are $\mathcal{NP}$-hard in the strong sense, which indicates the non-existence of efficient exact algorithms. This work proposes a guided and iterative heuristic search algorithm based on simulated annealing. The main part of this chapter was published in [HM12c] and [HM12a].

## 4.1 Introduction

In the computational complexity theory, $\mathcal{NP}$-hardness in the strong sense or strong $\mathcal{NP}$-hardness is a special case of $\mathcal{NP}$-hardness and indicates that the problem is even more tougher than $\mathcal{NP}$-hard problems in practice. In general, an optimization problem is said to be $\mathcal{NP}$-hard in the strong sense, if there exists a strongly $\mathcal{NP}$-complete problem that can be reduced to it in polynomial time. This is exactly the proof idea to show the strong $\mathcal{NP}$-hardness of both RTSC and RTMC problems. More specifically, the authors of [DA10] successfully reduced the 3-Partition problem, which is known to be strongly $\mathcal{NP}$-complete [GJ75], to the so-called RT-DPM problem, which is a simplified version of the RTSC problem by disabling the DVS capability.

According to another fundamental result in the complexity theory presented by Garey et al. [GJ78], the strongly $\mathcal{NP}$-hard problems, i.e., RTSC and RTMC, do not even admit a pseudo-polynomial time algorithm, whose runtime is polynomial in the numeric value rather than the size of the input. Note that a pseudo-polynomial time algorithm is probably the best algorithm one can expect for a $\mathcal{NP}$-hard problem.

By losing the hope for an exact pseudo-polynomial time algorithm, approximation algorithms sometimes seem to be very helpful, if one is satisfied with a solution that is ”good enough”. Unfortunately, it will be shown in the following text that no efficient approximation algorithm for the RTSC and RTMC problems exists, either. In order to measure the quality of a sub-optimal solution, the approximation ratio is introduced. Let $\varepsilon$ be a positive parameter, an algorithm for a minimization problem, without loss of generality, is said to be an $(1+\varepsilon)$-approximation algorithm, if the quality of the produced solution is within a factor $1+\varepsilon$ of the optimal solution. For instance, an $(1+\varepsilon)$-approximation algorithm for RTSC would find a speed assignment with the energy consumption over a hyper period being equal or less than the energy consumed by the optimal speed assignment with the factor $1+\varepsilon$. In practice, an approximation algorithm is said to be efficient, if its run-time is polynomial in both problem size and $1/\varepsilon$. This class is referred to as Fully Polynomial Time Approximation Scheme (FPTAS). Relating to pseudo-polynomial time algorithms, [Vaz02] has shown that a problem with a FPTAS must admit a pseudo-polynomial time algorithm (under some very weak restrictions). Since there exists no pseudo-polynomial time algorithm for RTSC and RTMC, no FPTAS can be found for them either. That is, there is no efficient approximation algorithm.

As a consequence, this work proposes to apply meta-heuristics to solve the problems, which tries to improve candidate solutions in an iterative manner. In general, a meta-heuristic defines high level strategies to efficiently guide the search process in the solution space. Some typical representatives are, for instance, Simulated Annealing (SA) [Kir+83] [Cer85], Evolutionary Algorithms (EA) [Bac96] [ES03] and Ant Colony Optimization algorithm (ACO) [Dor92]. Hereby, SA is adopted to solve the RTSC and RTMC problem because of two main reasons.

- It is very easy to implement, because SA is a single point search approach, as opposed to population based approaches. This saves a lot of run-time overhead, which will become more clear in Chapter 6.
- Its success has been shown in several other application fields, such as the layout optimization problem for integrated circuits design [VK83]. Some excellent experimental results have been reported in [Joh+91] as well, which indicates that SA could be a good alternative to handle $\mathcal{NP}$-hard problems by yielding acceptable performance in most cases.

Before the main algorithms for RTSC and RTMC are presented, an introduction of the generalized SA is given in this section. The SA algorithm was first introduced by Scott Kirkpatrick et al. in 1983 [Kir+83] and the basic concept is naturally inspired by annealing process in material science, which describes a metalworking process to alter the physical or chemical property of a material through heating or cooling. In general, the SA algorithm is a randomized local search heuristic. Let $\Omega$ denote the finite set of solutions, which is also the search space of the SA algorithm. Furthermore, let $J : \Omega \to \mathbb{R}$ be the real-valued cost function that defines the cost or the value for each solution in $\Omega$. Without loss of generality, hereby the minimization problem is considered, which indicates that the optimal solutions have the least cost. The set containing the optimal solutions is denoted by $\Omega^*$.

$$\Omega^* = \{\omega_i \in \Omega \mid \forall \omega_j \in \Omega : J(\omega_i) \leq J(\omega_j)\} \tag{4.1}$$

The SA algorithm initially starts with an arbitrary solution $\omega_{init} \in \Omega$ and iteratively explores the solution space by selecting one neighbor solution in each iteration. Let $N(\omega) \subseteq \Omega$ denote the set of all the neighbors of $\omega$. One assumes that the neighborhood structure is symmetric, i.e., $\forall \omega_1 \in \Omega$ and $\forall \omega_2 \in \Omega$, if $\omega_2 \in N(\omega_1)$ is true, then $\omega_1 \in N(\omega_2)$ holds as well. For simple applications, the probability of selecting a particular neighbor is usually defined as $1/|N(\omega)|$, where $|N(\omega)|$ denotes the size of the neighborhood. In other words, the neighbors in $N(\omega)$ are chosen following a uniform distribution. However, for the generalized SA the selection probability needs to be defined in a more general way. Thus, a probability function $Pr_{select} : \Omega^2 \to [0,1]$ is defined with the following properties. Let $\omega$ be an arbitrary element in $\Omega$.

$$Pr_{select}(\omega, \omega') > 0, \forall \omega' \in N(\omega) \tag{4.2}$$

$$Pr_{select}(\omega, \omega') = 0, \forall \omega' \notin N(\omega) \tag{4.3}$$

$$\sum_{\omega' \in N(\omega)} Pr_{select}(\omega, \omega') = 1 \tag{4.4}$$

Intuitively, if $\omega$ is the current solution, then a neighbor solution $\omega' \in N(\omega)$ is to be selected with the probability $Pr_{select}(\omega, \omega')$. At each iteration, once a neighbor solution is chosen, the cost of the current solution and the new solution is compared. If the new solution yields a lower cost, then it is definitely accepted and the new solution becomes the current solution. However, if the new solution is worse than the current solution, i.e., the current solution has a lower cost, then the new solution is accepted with a certain probability.

Afterwards, the SA algorithm goes to the next iteration and the process repeats until a termination criterion is reached. One unique feature of the SA algorithm is allowing the acceptance of worse solutions. This provides the advantage that the SA algorithm is able to escape from a local optimum, which is a solution with all its neighbor solutions being worse that itself.

Some theoretical results have even shown that if the iteration number is sufficiently large, the SA algorithm will eventually converge to a global optimum [MRS85].

More precisely, the acceptance probability is defined as a slowly and monotonically decreasing function of time, which exactly emulates the slow cooling behavior of the physical annealing process. Formally, a so-called cooling schedule is defined as a decreasing sequence of positive real numbers $(T_t)_{t\in\mathbb{N}}$ with $\lim_{t\to\infty} T_t = 0$ and $T_t$ is called the temperature at time or iteration $t$. There exist different forms to define a cooling schedule. This work applies the most popular cooling schedule (at least in theory [BT93]) which takes the form

$$T_t = \frac{d}{\log t} \tag{4.5}$$

where $d$ is a positive constant. Finally, the acceptance probability $Pr_{accept} : \Omega^2 \times \mathbb{N} \to [0,1]$ is defined.

$$Pr_{accept}(\omega, \omega', t) = \begin{cases} 1, & \text{if } J(\omega') < J(\omega) \\ exp(\frac{J(\omega)-J(\omega')}{T_t}), & \text{otherwise} \end{cases} \tag{4.6}$$

where $\omega$ and $\omega'$ are the current and new solution, respectively. Algorithm 1 shows a pseudo-implementation of the SA algorithm.

**Algorithm 1** Simulated Annealing Algorithm

```
Input: A solution space $\Omega$
Output: A solution $\omega_{curr}$
 1: Select $\omega_{init} \in \Omega$
 2: $\omega_{curr} \leftarrow \omega_{init}$, $i \leftarrow 1$
 3: while termination criterion is not met do
 4:     Select $\omega' \in N(\omega_{curr})$ according to $Pr_{select}(\omega_{curr}, \omega')$
 5:     Generate a random number $r \in [0,1]$
 6:     if $r \leq Pr_{accept}(\omega_{curr}, \omega', i)$ then
 7:         $\omega_{curr} \leftarrow \omega'$
 8:     end if
 9:     $i \leftarrow i+1$
10: end while
```

As a nature of meta-heuristics, the generalized SA algorithm is application-independent and thus can be applied to a large number of combinatorial optimization problems. However, this could also be a disadvantage, because it lacks problem specific knowledge when used in a concrete domain. Therefore, in what follows, the SA algorithm is customized toward the RTSC and

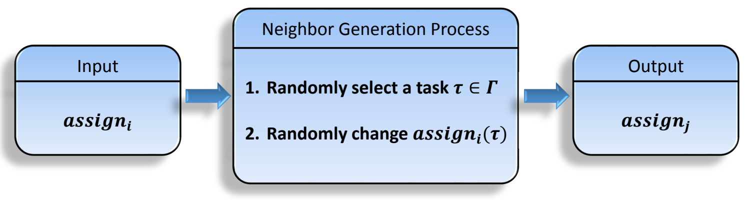

Input
$assign_i$
Neighbor Generation Process
1. Randomly select a task $\tau \in \Gamma$
2. Randomly change $assign_i(\tau)$
Output
$assign_j$

Figure 4.1: Neighbor generation process for the RTSC problem

RTMC problems, respectively. More concretely, specific rules are defined to guide neighbor selection at each iteration.

## 4.2 The HSASC Algorithm for RTSC

As a reminder, the RTSC problem addresses system-wide energy optimization for hard real-time tasks running on a single-core processor. The main objective is to find the optimal speed assignment *assign*, which ensures system schedulability while minimizing the system-wide energy consumption over one hyper period. Therefore, the solution space $\Omega_{RTSC}$ is defined as the set of all possible speed assignments.

$$\Omega_{RTSC} = \{assign_1, assign_2, ..., assign_{|\mathcal{S}|^n}\} \tag{4.7}$$

Clearly, the size of solution space is $|\mathcal{S}|^n$, where $|\mathcal{S}|$ is the number of available P-states and $n$ is the number of tasks. For instance, if a processor supports 5 operating speeds and 10 real-time tasks need to be scheduled, there exist $5^{10} = 9765625$ possible speed assignments in total. Moreover, two solutions $assign_i$ and $assign_j$ are said to be neighbors, i.e.,

$$(assign_i \in N(assign_j)) \wedge (assign_j \in N(assign_i)) \tag{4.8}$$

if they exactly differ in the speed assignment of one task, i.e.,

$$\exists \tau \in \Gamma : assign_i(\tau) \neq assign_j(\tau) \tag{4.9}$$

and

$$\forall \tau' \in \Gamma \setminus \{\tau\} : assign_i(\tau') = assign_j(\tau') \tag{4.10}$$

One of the most important procedures in the SA algorithm is neighbor selection. Naively, a uniform distribution can be applied to select neighbors, however, by considering problem specific information, some neighbors might be

| Task | $\tau_1$ | $\tau_2$ | $\tau_3$ | $\tau_4$ | $\tau_5$ |
|---|---|---|---|---|---|
| *SC* | $S_3$ | $S_3$ | $S_3$ | $S_3$ | $S_3$ |
| *assign* | $S_1$ | $S_3$ | $S_3$ | $S_2$ | $S_1$ |
| *pen* | 1.8 | 1 | 1 | 1.3 | 1.8 |
| *prob* | 0.26 | 0.14 | 0.14 | 0.20 | 0.26 |

Table 4.1: An example computation of the task penalty value and the task selection probability

more superior than the others in terms of energy consumption. For the RTSC problem, this process is performed by selecting a neighbor in two steps as shown in Figure 4.1. In the first step a task is selected, for which its frequency is to be changed in the second step. For the first step, each task is associated with a so-called penalty value $pen(\tau_i)$, which is defined in (4.11).

$$\forall \tau_i \in \Gamma : pen(\tau_i) = \lambda_1 \cdot |F(assign(\tau_i)) - F(SC(\tau_i))| + \lambda_2 \qquad (4.11)$$

Intuitively, the penalty value of a task shows the distance between the currently assigned P-state and the P-state with critical speed. By definition, the critical speed of a task is the optimal speed when its AEC is concerned. Obviously, if the task is not running at the optimal speed, then there is some wasted energy. Therefore, the task is to be penalized according to its difference to the optimal speed. The parameter $\lambda_1$ is a coefficient to adjust the weight of the speed difference and $\lambda_2$ is a technical parameter preventing the penalty value from being zero. After establishing the penalty value, the task selection probability $prob(\tau_i)$ in the first step is defined.

$$\forall \tau_i \in \Gamma : prob(\tau_i) = \frac{pen(\tau_i)}{\sum\limits_{\tau_j \in \Gamma} pen(\tau_j)} \qquad (4.12)$$

It is not hard to see that the more penalized task is to be selected more likely, because in this way, hopefully, its change is more beneficial with regard to energy saving. Table 4.1 shows a simple example with 5 real-time tasks. It is assumed that the processor supports 3 P-states with $F(S_1) = 1$, $F(S_2) = 0.5$ and $F(S_3) = 0.2$. The critical speed for each task is assumed to be computed and shown in the second row. Though all the tasks have the same critical speed here, it may not be the case for other examples. Furthermore, the example assumes *assign* as the current solution shown in the third row of the table. The corresponding task penalty values can be easily obtained by applying (4.11), provided that $\lambda_1$ and $\lambda_2$ are set to be 1. In accordance with (4.12), the calculated selection probability for each task is shown in the fifth row.

After a task $\tau_i$ is selected, a new speed has to be assigned to it. Hereby the critical speed is selected with the probability 0.5 and the remaining probability is uniformly distributed to the other available speeds. Algorithm 2 shows an overview of the proposed heuristic search algorithm. The initial solution is simply generated by assigning all the tasks with the maximal speed.

**Algorithm 2** Heuristic Search Algorithm for RTSC (HSASC)

**Input:** A solution space $\Omega = \{assign_1, assign_2, ..., assign_{|\mathcal{S}|^n}\}$
**Output:** A solution $assign_{best}$

```
1:  Generate $assign_{init}$ with $\forall\tau \in \Gamma : assign_{init}(\tau) = S_1$
2:  $assign_{best} \leftarrow assign_{init}$, $assign_{curr} \leftarrow assign_{init}$, $i \leftarrow 1$
3:  while termination criterion is not met do
4:      Select a $\tau$ according to the selection probability $prob$ (4.12)
5:      Generate $assign'$ by changing $assign_{curr}(\tau)$
6:      Generate a random number $r \in [0,1]$
7:      if $r \leq Pr_{accept}(assign_{curr}, assign', i)$ then
8:          $assign_{curr} \leftarrow assign'$
9:          if $J(assign_{curr}) < J(assign_{best})$ then
10:             $assign_{best} \leftarrow assign_{curr}$
11:         end if
12:     end if
13:     $i \leftarrow i + 1$
14: end while
```

In Algorithm 2, the initial and current solution are denoted by $assign_{init}$ and $assign_{curr}$, respectively. $assign'$ denotes the generated neighbor solution in each iteration. Except the neighbor selection process, another important difference between the HSASC Algorithm and the generalized SA algorithm (Algorithm 1) is that the HSASC algorithm always remembers the best solution, denoted by $assign_{best}$, ever found and produces it as the final output. This modification further improves the performance of the original SA algorithm, which outputs the current solution.

Moreover, $J(assign)$ expresses the cost or value of the solution $assign$, which is the system-wide energy consumption over a hyper period. If a speed assignment results in an invalid solution, i.e., the real-time system is not feasible under the speed assignment, the HSASC algorithm assumes that $J(assign) = +\infty$. As a result, the unschedulable solutions will never be accepted. In general, the evaluation of a solution is a non-trivial job. At this moment, one assumes that the value of a solution can be obtained in some way. However, more details are explained in Chapter 6.

Table 4.2 shows a possible series of solutions that are visited by the HSASC algorithm for the example shown in Table 4.1.

| Task | $\tau_1$ | $\tau_2$ | $\tau_3$ | $\tau_4$ | $\tau_5$ |
|---|---|---|---|---|---|
| 1st *assign* | $S_1$ | $S_1$ | $S_1$ | $S_1$ | $S_1$ |
| 2nd *assign* | $S_2$ | $S_1$ | $S_1$ | $S_1$ | $S_1$ |
| 3rd *assign* | $S_2$ | $S_3$ | $S_1$ | $S_1$ | $S_1$ |
| 4th *assign* | $S_3$ | $S_3$ | $S_1$ | $S_1$ | $S_1$ |
| 5th *assign* | $S_3$ | $S_3$ | $S_1$ | $S_3$ | $S_1$ |
| 6th *assign* | $S_3$ | $S_3$ | $S_2$ | $S_3$ | $S_1$ |
| 7th *assign* | $S_3$ | $S_3$ | $S_2$ | $S_3$ | $S_2$ |

Table 4.2: A series of possible solutions

## 4.3 The HSAMC Algorithm for RTMC

The RTMC problem deals with the energy optimization problem on cluster-based multi-core processor platforms. The main objective is to find a task partition and a speed assignment. Therefore, the solution space $\Omega_{RTMC}$ is defined as follows.

$$\Omega_{RTMC} = \{(alloc, assign)_1, (alloc, assign)_2, ..., (alloc, assign)_{(\sum_{O_i} |\mathcal{S}^i|)^n}\} \tag{4.13}$$

The size of solution space is $(\sum_{O_i} |\mathcal{S}^i|)^n$, where $|\mathcal{S}^i|$ is the number of available P-states on $O_i$. $n$ is the number of tasks. For instance, one considers a quad-core processor, where each processor core supports 5 P-states. If 10 real-time tasks need to be scheduled, the number of possible solutions are $(4 \cdot 5)^{10} \approx 10^{13}$. Furthermore, two solutions are said to be in a neighborhood, if they differ in the configuration of one task, i.e.,

$$\exists \tau \in \Gamma : assign_i(\tau) \neq assign_j(\tau) \vee alloc_i(\tau) \neq alloc_j(\tau) \tag{4.14}$$

and

$$\forall \tau' \in \Gamma \setminus \{\tau\} : assign_i(\tau') = assign_j(\tau') \wedge alloc_i(\tau') = alloc_j(\tau') \tag{4.15}$$

The neighbor selection process is similar to the case of RTSC, however, there is one more step for the RTMC problem. More specifically, a neighbor of the current solution is generated in three steps: i) select a task, ii) change its processor core allocation and iii) change its speed assignment. Figure 4.2 illustrates the generation process. Note that the other possibility is to first make reassignment and perform reallocation afterward. In this case, the

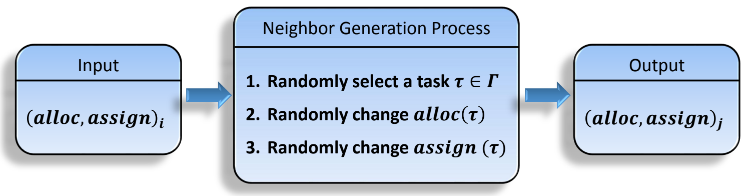

Input
$(alloc, assign)_i$
Neighbor Generation Process
1. Randomly select a task $\tau \in \Gamma$
2. Randomly change $alloc(\tau)$
3. Randomly change $assign\ (\tau)$
Output
$(alloc, assign)_j$

Figure 4.2: Neighbor generation process for RTMC

reallocation step may be dependent on the result of the reassignment step, if a multi-core processor with heterogeneous cores in terms of power model is considered. For instance, if a processor contains two cores $O_1$ and $O_2$ grouped into two clusters and $O_1$ and $O_2$ support different range of speeds, the reassignment step may choose one speed that is supported by only one of the processor cores, e.g., $O_2$. As a result, the reallocation step may only select $O_2$ as the target. In this dissertation, the focus is on the first option that the core reallocation is made before the speed reassignment. The second possibility, however, will be further discussed in Chapter 9 as part of the future work.

For task selection (the first step) the guidance is defined based on the heuristic information extracted from the current solution. More precisely, each task $\tau_i$ is associated with a penalty value denoted by $pen(\tau_i)$. On the analogy to the RTSC problem, the penalty value of a task indicates the wasted energy by the task. The higher the penalty value, the more possibly the corresponding task is to be selected for a configuration change. $pen(\tau_i)$ is formally computed by means of (4.16), which is composed of three parts (on the right side of the equation).

$$\begin{aligned} pen(\tau_i) \quad = \quad & \lambda_1 \cdot |F(assign(\tau_i)) - F(SC^j(\tau_i))| + \\ & \lambda_2 \cdot t_{unbalanced}(\tau_i) + \lambda_3 \end{aligned} \tag{4.16}$$

Like the penalty function definition for the RTSC problem, the first part expresses the wasted active energy during task execution. Note that, hereby it is assumed that $\tau_i$ is running on $O_j$. Due to the speed coordination constraint of cluster-based multi-core processor platforms, an additional term is added into the penalty function. Namely, the second part describes the unbalanced task execution in a cluster. More specifically, $t_{unbalanced}(\tau_i)$ denotes the time inside a hyper period, where a processor core $O$ executes $\tau_i$ at $assign(\tau_i)$ and at least one of the other cores $O'$ in the same cluster is enforced to operate at the same speed. In other words, by disregarding the processor core $O$, the active task on $O'$ should have run at a lower speed than $assign(\tau_i)$. Obviously, the task with larger $t_{unbalanced}(\tau_i)$ should be more penalized.

Figure 4.3 shows an example of task execution with two processor cores in

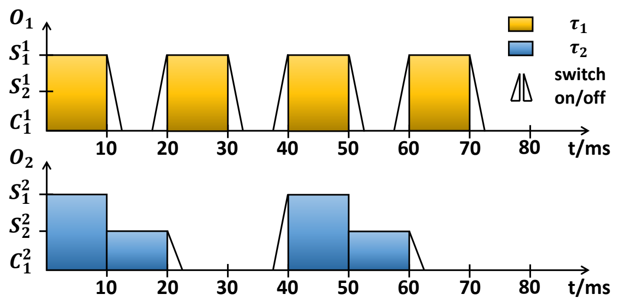


Figure 4.3: An example task execution on a dual-core processor

| Task | $W$ (ms) | $T$ (ms) | $SC^1$ | $SC^2$ |
|---|---|---|---|---|
| $\tau_1$ | 10 | 20 | $S_2^1$ | $S_2^2$ |
| $\tau_2$ | 15 | 40 | $S_2^1$ | $S_2^2$ |

Table 4.3: The task specification

one cluster. In this example, there are two P-states ($F(S_1^1) = F(S_1^2) = 1$ and $F(S_2^1) = F(S_2^2) = 0.5$) available and EDF schedule is assumed to be used. The task specification is given in Table 4.3. The example further assumes that $\tau_1$ runs on $O_1$ and $\tau_2$ on $O_2$. $S_1^1$ is assigned to $\tau_1$ and $S_2^2$ to $\tau_2$, respectively. Due to the speed coordination, task $\tau_2$ is running at a higher speed than its originally assigned speed during the time interval $[0\text{ ms}, 10\text{ ms}]$. In this case $t_{unbalanced}(\tau_1)$ is equal to 10 ms. The constants $\lambda_1$ and $\lambda_2$ in (4.16) serve as coefficients to adjust the impact of the first and the second part, respectively. The constant $\lambda_3$ is a technical parameter in order to prevent the penalty value from being zero. By assuming $\lambda_1 = \lambda_2 = \lambda_3 = 1$, the task penalty values in the example can be computed as follows.

$$pen(\tau_1) = |F(S_1^1) - F(S_2^1)| + 10 + 1 = 11.5 \tag{4.17}$$

$$pen(\tau_2) = |F(S_2^2) - F(S_2^2)| + 0 + 1 = 1 \tag{4.18}$$

Based on the penalty values, the selection probability $prob(\tau_i)$ of each task can be derived.

$$prob(\tau_i) = \frac{pen(\tau_i)}{\sum\limits_{\tau_j \in \Gamma} pen(\tau_j)} \tag{4.19}$$

After a task $\tau_k$ is selected, in the second step the task is to be reallocated to a new core. Hereby it is quite reasonable to balance processor load and avoid generating invalid solutions, where system schedulability is not ensured. For this purpose each processor core $O_j$ is associated with a reward value $rew(O_j)$, which is simply the available utilization that is still free to be used. Formally the reward function $rew(O_j)$ is defined in (4.20).

$$rew(O_j) = \begin{cases} U_B^j - U(O_j), & \text{if } alloc(\tau_k) \neq O_j \\ U_B^j - U(O_j) + U(\tau_k), & \text{otherwise} \end{cases} \tag{4.20}$$

$U_B^j$ is the utilization upper bound on the processor core $O_j$ to ensure schedulability. For instance, $U_B^j$ is 1 if the EDF scheduling algorithm is used. $U_B^j$ is 0.69, if the RM algorithm is used. Moreover, $U(O_j)$ is the current processor utilization on the core $O_j$ and $U(\tau_k)$ is the utilization of the task $\tau_k$, assuming that $\tau_k$ is selected in the first step. In the example shown in Figure 4.3, if the task $\tau_1$ is selected and needs to be reallocated, the reward value for each processor core can be computed by

$$\begin{aligned} rew(O_1) &= 1 - \frac{W(\tau_1) \cdot F(S_1^1)}{F(S_1^1) \cdot T(\tau_1)} + \frac{W(\tau_1) \cdot F(S_1^1)}{F(S_1^1) \cdot T(\tau_1)} \\ &= 1 - \frac{10}{20} + \frac{10}{20} \\ &= 1 \end{aligned} \tag{4.21}$$

and

$$\begin{aligned} rew(O_2) &= 1 - \frac{W(\tau_2) \cdot F(S_1^2)}{F(S_2^2) \cdot T(\tau_2)} \\ &= 1 - \frac{30}{40} \\ &= 0.25 \end{aligned} \tag{4.22}$$

Analogous to task selection, the selection probability $prob(O_j)$ of each core can be derived on the basis of the reward values.

$$prob(O_j) = \frac{rew(O_j)}{\sum\limits_{O_i \in \mathcal{O}} rew(O_i)} \tag{4.23}$$

Intuitively, the processor core with more free utilization will be selected more likely.

Finally, in the third step a new speed is to be assigned to the selected task. Hereby the critical speed is selected with the probability 0.5 and the remaining probability is uniformly distributed to the other available speeds. Algorithm 3 shows an overview of the proposed heuristic search algorithm. The initial solution is simply generated by means of the worst fit strategy [DB11] to partition the tasks and afterward assigning all the tasks with the maximal speed.

**Algorithm 3** Heuristic Search Algorithm for RTMC (HSAMC)

**Input:** The solution space $\Omega_{RTMC}$
**Output:** A solution $(alloc, assign)_{best}$

```
1:  Generate $(alloc, assign)_{init}$
2:  $(alloc, assign)_{best} \leftarrow (alloc, assign)_{init}$
3:  $(alloc, assign)_{curr} \leftarrow (alloc, assign)_{init}$
4:  $i \leftarrow 1$
5:  while termination criterion is not met do
6:      Select a $\tau$ according to the penalty values
7:      Reallocate $\tau$ according to the reward values
8:      Reassign $\tau$ with a new speed to obtain $(alloc, assign)'$
9:      Generate a random number $r \in [0,1]$
10:     if $r \leq Pr_{accept}((alloc, assign)_{curr}, (alloc, assign)', i)$ then
11:         $(alloc, assign)_{curr} \leftarrow (alloc, assign)'$
12:         if $J((alloc, assign)_{curr}) < J((alloc, assign)_{best})$ then
13:             $(alloc, assign)_{best} \leftarrow (alloc, assign)_{curr}$
14:         end if
15:     end if
16:     $i \leftarrow i + 1$
17: end while
```

The HSAMC algorithm returns the best solution it has ever visited, which results in a better performance than the original version of SA. Similar to the HSASC algorithm, the cost or value of a solution is defined as the energy consumption over one hyper period. If the solution is not able to guarantee system schedulability, its cost is set to infinity. Hereby the cost of a solution is assumed to be obtained in some way. More details are explained in Chapter 6.

Table 4.4 shows a possible sequence of solutions that are visited by the HSAMC algorithm for the example shown in Figure 4.3.

## 4.4 Chapter Summary

This chapter first gave an review of the RTSC and RTMC problems in terms of complexity. Unfortunately, both problems are proven to be $\mathcal{NP}$-hard in the strong sense. As a result, there is no efficient algorithm available. Subsequently, two heuristic search algorithms based on simulated annealing for

| Task | $\tau_1$ | $\tau_2$ |
|---|---|---|
| 1st solution | $(O_1, S_1)$ | $(O_2, S_1)$ |
| 2nd solution | $(O_1, S_1)$ | $(O_2, S_2)$ |
| 3rd solution | $(O_2, S_2)$ | $(O_2, S_2)$ |
| 4th solution | $(O_1, S_2)$ | $(O_2, S_2)$ |
| 5th solution | $(O_1, S_2)$ | $(O_1, S_1)$ |

Table 4.4: A series of possible solutions

RTSC and RTMC are proposed, which are the main contribution of this chapter.

# Chapter 5

# Run-Time Behavior Analysis

The previous chapter introduced the heuristic algorithms for energy optimization problem in the context of hard real-time systems. The guided search approach is iterative and based on the simulated annealing algorithm. Though it may work well in most cases, there is no guarantee that the heuristic algorithm will produce a globally optimal solution. There is even no information about the quality of the output solution. In this chapter, an efficient mechanism is proposed to efficiently analyze and predict the performance of the algorithm. A termination criterion is derived as well. The main part of this chapter is published in [HM13a].

## 5.1 Introduction

Due to the hypothetical assumption that there exists no efficient algorithm for $\mathcal{NP}$-hard or strongly $\mathcal{NP}$-hard problems, meta-heuristics become quite popular and are widely used in practice. Even though in most cases they are able to produce excellent results within reasonable time, there is no assurance of finding a global optimum at the end of the algorithm. One has to be satisfied with the sub-optimal performance. Unfortunately, the SA based HSASC and HSAMC algorithms can not escape this fate either. In general, there are two important questions that need to be answered.

- Q1: What is the termination criterion of the algorithm or after how many iterations the algorithm should be stopped?
- Q2: What is the quality of the final solution? If it is not globally optimal, then how far away it is from a globally optimal solution?

Obviously, these questions are not trivial to answer. In the context of SA, some efforts have been carried out in the direction of convergence theory, which tries to prove that the SA algorithm will eventually find a global optimum, if the run-time of the algorithm is sufficiently large. Mitra et al. [MRS85] [DS86] presented a theoretical analysis of the SA algorithm by modeling it as a time-inhomogeneous Markov chain. The convergence result is then proven by showing the strong ergodic property of the Markov chain. More specifically, the probability distribution of solutions $\pi_t : \Omega \to [0,1]$ depending on the iteration number $t$ is defined to describe the probability of each solution that might be produced as the final solution if the SA algorithm terminates after $t$ iterations. At the beginning, $\pi_0$ is a uniform distribution indicating the initial solution is arbitrarily chosen. The goal is to prove that the SA algorithm reaches a so-called quasi-stationary probability distribution $\pi^*$, as $t \to +\infty$.

$$\pi^*(\omega) = \begin{cases} \frac{1}{|\Omega^*|}, & \text{if } \omega \in \Omega^* \\ 0, & \text{otherwise} \end{cases} \tag{5.1}$$

Note that $\Omega^*$ denotes the set of the optimal solutions. In addition to the convergence result, [MRS85] gave an asymptotic performance analysis of the finite time behavior of the SA algorithm as well, which bounds the distance between $\pi_t$ and $\pi^*$ depending on $t$. This effectively estimates the convergence rate. Nolte et al. [NS00] proposed an alternative means by using rapidly mixing Markov chains and proved the asymptotic convergence result as well. They demonstrated the analysis on a well-known $\mathcal{NP}$-hard problem, the traveling salesman problem [MT13].

Sasaki et al. [SH88] performed a time complexity analysis by applying the SA algorithm to the maximum matching problem, also known as maximum carnality matching. In fact, this problem is in the complexity class $\mathcal{P}$, because there is a known polynomial time algorithm called Edmonds's algorithm [Edm65]. Clearly, no one would ever use the SA algorithm to solve this problem in practice. However, [SH88] gave some insights on the analysis technique to understand the run-time behavior of the SA algorithm. One important result shows that a special form of the generalized SA algorithm, where the temperature is a constant instead of being dependent on time, is able to produce near optimum solutions in polynomial average time. The research studies [Des99] and [Des92] applied an eigenvalue-based approach to analyze the time complexity of the same problem. Their result is based on the general cooling schedule and shows that the SA algorithm effectively serves as a polynomial randomized approximation algorithm.

The author of [Loc01] investigated the performance of the SA algorithm for continuous optimization problems. The study proved the convergence of the algorithm and gave an upper bound of the expected run-time to find an optimal solution within required accuracy.

Unlike the above mentioned research work, the authors of [OJ02], [JHM06], [Nik+11] and [Jac+05] introduced a semi-analytic method to assess the finite time performance of local search algorithms. They derived an upper and lower bound of the expected time to hit a so-called $\beta$-acceptable solution, which is a near optimum solution with approximation ratio $\beta$. In order to use their results, some measurement data have to be collected beforehand.

For additional information, survey articles [HJJ03] and [NJ10] give excellent overview of the SA algorithm from both the theoretical and practical view of point.

As a short summary, there exist a large number of research work regarding the performance analysis of the SA algorithm. However, most of them are only relevant in theory. For instance, the convergence result shows that the algorithm is able find a global optimum, if the number of iterations is infinity. In practice, infinite time is clearly not acceptable. Several studies give the asymptotic analysis of finite time behavior, however, the results are often related to a specific optimization problem and in some cases the problem is even in the complexity class $\mathcal{P}$. No existing results can be directly applied to exactly answer the questions Q1 and Q2. Therefore, the remaining of this chapter tries to propose a more practical technique to analyze the performance of the SA and SA-based algorithm. The key idea is inspired by the work [RWep] proposed by Raman et al., who used exponential regression technique to simulate and analyze the finite time behavior of the algorithm. This dissertation further improves their approach by allowing it to be preformed at run-time. The regression analysis is a mathematical method widely applied in statistics to estimate the behavior of data. It often serves as a tool to understand the relationship between dependent variables. An introduction can be found in [GC04].

## 5.2 Run-Time Behavior Analysis through Exponential Regression

In order to better understand the run-time behavior of the proposed algorithms, several experiments through simulation are firstly made. The simulation framework is built on the basis of a SystemC RTOS library [ZMG09], which provides the basic functions for simulating a set of real-time tasks under well-known scheduling algorithms, such as EDF and RM. The library is extended by additional support for DPM and DVS capabilities. A more detailed introduction is given in Chapter 8.

The RTSC problem is addressed in the first experiment, where the virtual platform is composed of an Intel XScale processor [Xu+04] and five I/O devices [CG06] including a MaxStream Wireless Module, an IBM Microdrive,

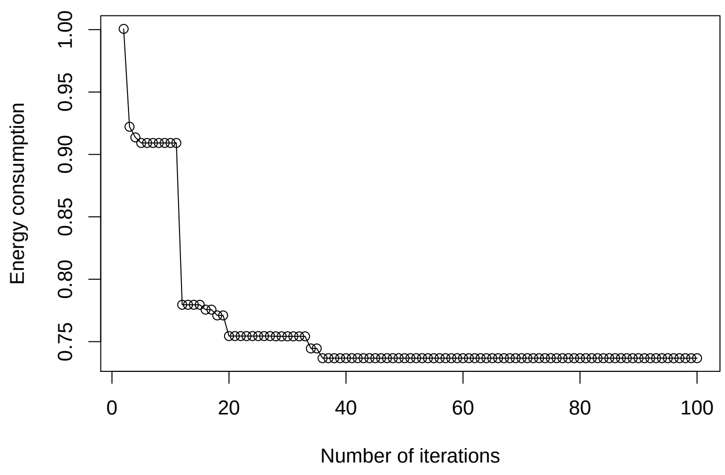


Figure 5.1: Run-time behavior of the HSASC algorithm (example 1)

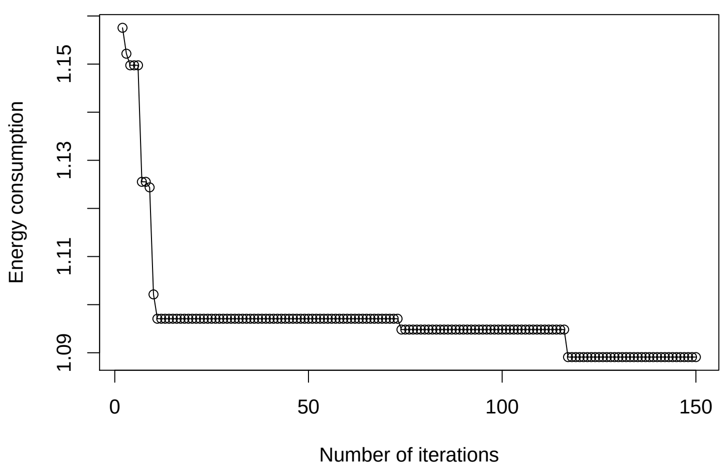


Figure 5.2: Run-time behavior of the HSAMC algorithm (example 2)

an SST Flash, a SimpleTech Flash Card and a Realtek Ethernet Chip. Several synthetic task sets are randomly generated. For demonstration purpose, Figure 5.1 shows the recorded simulation results by applying the HSASC algorithm on one of the generated task sets. The x-axis shows the number of iterations and the y-axis is the value of the best solution found so far.

In the second experiment, the RTMC problem is investigated and the virtual platform is built by multiplying the Intel XScale processor model. More concretely, a quad-core processor is used in the simulation and each processor core takes the same power model as the Intel XScale processor. In addition, the processor cores are divided into two clusters of the same size. Figure 5.2 illustrates the simulation result by applying the HSAMC algorithm on one randomly generated task set. The meaning of the x- and y-axis are the same as in Figure 5.1.

There are two major observations that can be obtained from these two figures.

1. The more iterations spent in the algorithm, the better solution can be obtained.
2. The value of the best solution exponentially decreases as the algorithm proceeds.

By the other experiments the same trend is observed as well. Therefore, it is reasonable to conclude that the solution value is an exponentially decreasing function of the number of iterations.

The core idea is then to use exponential regression technique at run-time to simulate the behavior of the algorithm and try to predict the quality of solutions. Obviously, the accuracy of regression gets better while the algorithm advances and the number of observed data points increases. The regression model takes the form,

$$y_i = ae^{bx_i} + c, \quad i = 1, \ldots, l \tag{5.2}$$

where $y_i$ and $x_i$ are the response variable and the input variable, respectively. The regression parameters $a$, $b$ and $c$ are to be determined by curve fitting to a series of observed data points $\{y_i, x_i\}_{i=1}^{l}$. Note that $y_i$ indicates the value of the best solution found by the algorithm until the $i$-th iteration and $x_i$ is the iterations spent so far, i.e.,

$$\forall i : 1 \leq i \leq l, x_i = i \tag{5.3}$$

The parameter $c$ is in fact the value of the optimal solution, to which the regression function eventually converges. Unfortunately, solving the exponential regression is not a trivial work and traditional approaches, such as the Gauss-Newton algorithm [BW08], are often very time consuming. However, there exists linearization technique transforming the exponential regression problem to the linear regression problem, provided that the parameter $c$ can be eliminated. Thus the exponential regression model is reformulated into the following function:

$$\begin{aligned} y'_i &= y_{i-1} - y_i \\ &= ae^{bx_{i-1}} + c - \left(ae^{bx_i} + c\right) \;\; i = 2, \ldots, l \end{aligned} \tag{5.4}$$

By applying (5.3), it is obvious that,

$$x_{i-1} = i - 1 = x_i - 1 \tag{5.5}$$

Thus (5.4) can be further transformed as follows.

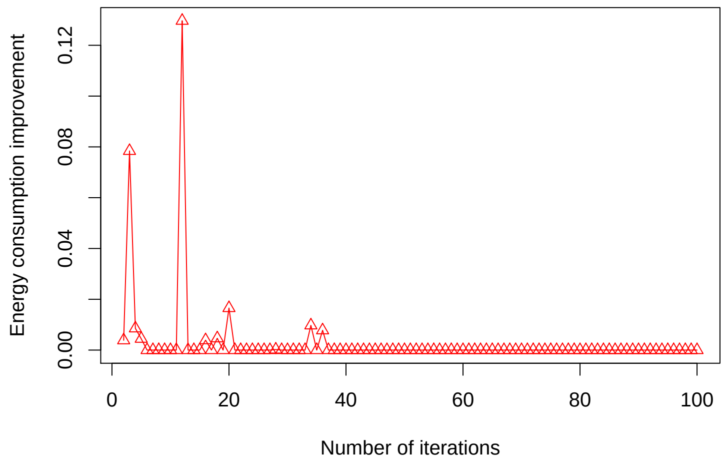


Figure 5.3: Run-time behavior of the solution value improvement (example 1)

$$\begin{aligned} y'_i &= ae^{b(x_i-1)} + c - (ae^{bx_i} + c) \\ &= (ae^{-b} - a)e^{bx_i}, \quad i = 2, \ldots, l \end{aligned} \tag{5.6}$$

Intuitively, $y'_i$ expresses the value improvement at the $i$-th iteration. Figure 5.3 and 5.4 illustrate its run-time behavior based on the previous examples. The behavior of improvements actually reflects the gradient of the regression function and is an exponentially decreasing function of the iteration number. A larger improvement represents a larger gradient value. On the contrary, smaller improvements indicate smaller gradient values. As the algorithm proceeds, the value improvement clearly becomes smaller and smaller and eventually converges to zero. Note that this convergence implies the convergence of the regression function (5.2) to the optimal solution.

By looking at (5.6), since the offset parameter $c$ disappears, it can be linearized to a linear regression model shown in (5.7). Hereby the nature logarithm is applied.

$$\begin{aligned} ln(y'_i) &= ln((ae^{-b} - a)e^{bx_i}) \\ &= ln(ae^{-b} - a) + ln(e^{bx_i}) \\ &= ln(ae^{-b} - a) + bx_i, \quad i = 2, \ldots, l \end{aligned} \tag{5.7}$$

Let $Y_i = ln(y'_i)$ and $X_i = x_i$, the final linear regression model is obtained in (5.8), where the slope and offset are $b$ and $ln(ae^{-b} - a)$, respectively.

$$Y_i = bX_i + ln(ae^{-b} - a), \quad i = 2, \ldots, l \tag{5.8}$$

Now one can apply the well-known *Least-Squares Estimation* (LSE) tech-

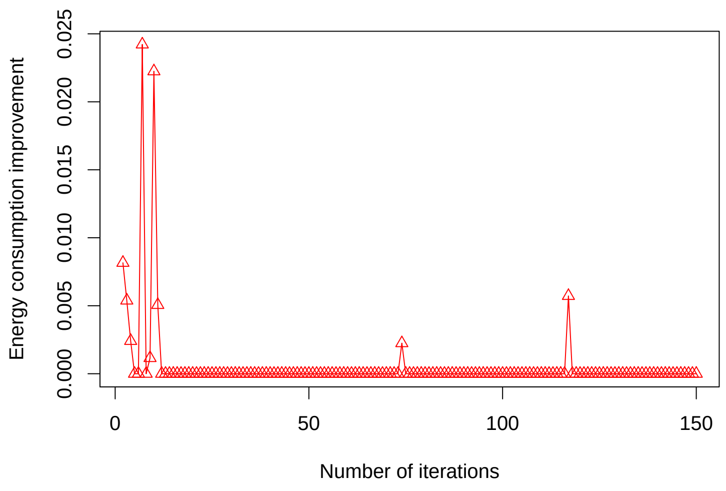


Figure 5.4: Run-time behavior of the solution value improvement (example 2)

nique [MPV12] by means of minimizing the sum of squared residuals to determine the regression parameters.

$$b = \frac{(n-1)\sum_{i=2}^{n} X_i Y_i - \sum_{i=2}^{n} X_i \sum_{i=2}^{n} Y_i}{(n-1)\sum_{i=2}^{n} X_i^2 - (\sum_{i=2}^{n} X_i)^2} \tag{5.9}$$

$$ln(a(e^{-b}-1)) = \frac{\sum_{i=2}^{n} Y_i \sum_{i=2}^{n} X_i^2 - \sum_{i=2}^{n} X_i \sum_{i=2}^{n} X_i Y_i}{(n-1)\sum_{i=2}^{n} X_i^2 - (\sum_{i=2}^{n} X_i)^2} \tag{5.10}$$

Note that $(Y_i, X_i)$ is the observed data point at the $i$-th iteration. In order to obtain the parameters $a$ and $b$, mainly four expressions on the right side of the equations (5.9) and (5.10) are to be determined: $\sum_{i=2}^{n} X_i$, $\sum_{i=2}^{n} X_i^2$, $\sum_{i=2}^{n} Y_i$ and $\sum_{i=2}^{n} X_i Y_i$. Since the key idea is to perform regression while the algorithm is running, these four expressions need not to be calculated from scratch every time, but rather only have to be cumulatively updated at each iteration. Algorithm 4 shows the procedure for updating the regression parameters at each iteration of the HSASC and HSAMC algorithms.

$\delta$ in Algorithm 4 denotes the value improvement at the current iteration. According to (5.7), $Y_i = ln(y_i') = ln(y_{i-1} - y_i) = ln(\delta)$. The variable $\bar{X}$ remembers the current iteration number.

There is still one problem in the calculation of the regression parameters. Namely, the value of $\delta$ is not allowed to be zero, as the term $ln(0)$ is undefined. In order to avoid this behavior, this work proposes to use the smoothing technique to forecast such ”unknown” data. Furthermore, there is also a positive side effect, because the regression technique now works on the smoothed data rather than the original one. The ”noise” data are often smoothed out.

**Algorithm 4** The Regression Procedure

```
1: $\sum X \leftarrow 0, \sum X^2 \leftarrow 0, \sum Y \leftarrow 0, \sum XY \leftarrow 0$ and $\bar{X} \leftarrow 1$
2: for each iteration do
3:     $\bar{X} \leftarrow \bar{X} + 1$
4:     $\sum X \leftarrow \sum X + \bar{X}$
5:     $\sum X^2 \leftarrow \sum X^2 + \bar{X}^2$
6:     $\sum Y \leftarrow \sum Y + ln(\delta)$
7:     $\sum XY \leftarrow \sum XY + \bar{X} \cdot ln(\delta)$
8:     Compute $a$ and $b$ according to (5.10) and (5.9)
9: end for
```

Hereby, the exponential smoothing approach is applied which tries to weight the past data with regard to their time stamp, i.e., the older the data, the less the weight. The smoothing equation is shown in (5.11) where $\delta$ and $\hat{\delta}$ are the originally observed data and the smoothed data, respectively.

$$\hat{\delta}_i = \begin{cases} \delta_i, & \text{if } \delta_i > 0 \\ \alpha \cdot \delta_i + (1-\alpha) \cdot \hat{\delta}_{i-1}, & \text{otherwise} \end{cases} \tag{5.11}$$

$\alpha$ is the smoothing constant and the less the value, the more the data set is smoothed. Based on the previous examples, Figure 5.5 and 5.6 show the comparison between the original data and the smoothed data with $\alpha = 0.1$. Clearly all the "zero" data are replaced by "non-zero" data. Moreover, the smoothing constant $\alpha$ is a key factor to steer the convergence speed of $\hat{\delta}$ toward zero. The larger the $\alpha$, the faster the curve of $\hat{\delta}$ approaches zero. Since the value improvement is an indicator of the gradient, the fast convergence consequently implies a fast termination of the algorithm. Obviously, fast termination has its price, because some local optima may be mistakenly recognized as global optima and reported as the final output of the algorithm. The right choice of $\alpha$ has a significant impact on the performance analysis result.

Figure 5.7 shows the differently smoothed values with different smoothing constants. It is not hard to observe that a larger $\alpha$ results in a faster convergence to 0. The blue line ($\alpha = 0.5$) converges to 0 at the fastest speed and the purple line with $\alpha = 0.01$ behaves the opposite. However, by setting $\alpha = 0.5$ the algorithm will miss the improvement marked by the red circle, because it would have already terminated before. In other words, using a larger $\alpha$ will produce a relatively worse solution. Therefore, $\alpha$ is a trade-off factor that represents a compromise between the convergence speed and the solution quality.

In order to find the best choice of the smoothing constant, a mechanism is

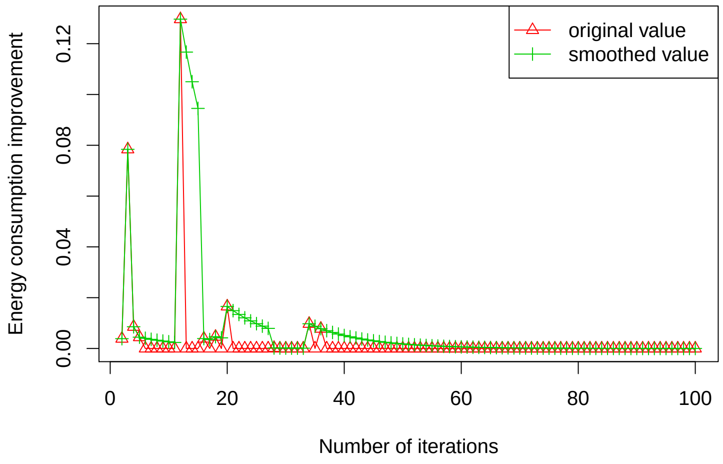


Figure 5.5: The original data vs. the smoothed data (example 1)

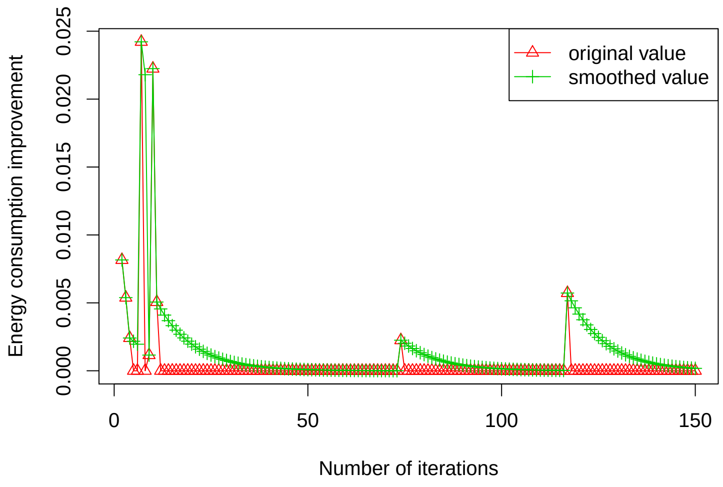


Figure 5.6: The original data vs. the smoothed data (example 2)

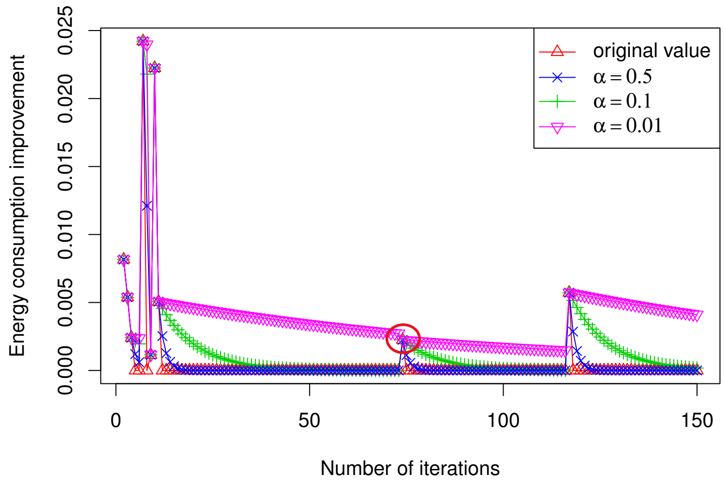


Figure 5.7: The impact of different smoothing constants (example 2)

proposed to dynamically adjust $\alpha$ at run-time using heuristic information. More specifically, $\alpha$ is updated at each iteration of the algorithm by means of (5.12).

$$\alpha = \frac{1}{n \cdot o \cdot \exp(\frac{\sum_{i=1}^{n} |F(assign(\tau_i)) - F(SC(\tau_i))|}{n})} \tag{5.12}$$

Hereby $n$ is task number and $o$ is the number of processor cores. Intuitively, $\alpha$ is mainly derived from the average difference between the assigned speed and the critical speed. As the critical speed admits the optimal task AEC, it is an important indicator of solution quality. In other words, if the average difference is fairly large, then it indicates that the solution quality is relatively poor. Thus the convergence speed needs to be slowed down with a smaller $\alpha$. On the other hand, the equation (5.12) yields a larger $\alpha$, if the assigned speeds are equal to or close to the respective critical speeds. Moreover, $\alpha$ here is also dependent on the number of cores $o$ and the number of tasks $n$. The main idea behind this is to slow down the termination speed when these numbers are high, because in this situation the solution space is rather large and more time should be spent for solution search. Note that (5.12) obviously ensures $0 \leq \alpha \leq 1$.

To briefly sum up, this section introduced a regression based technique to simulate and predict the run-time behavior of the algorithms at run-time. An exponential regression model is used as basis and the regression parameters $a$ and $b$ can be estimated at each iteration. In what follows, $a$ and $b$ will be applied to estimate the quality of solutions and finally derive the termination criterion for the HSASC and HSAMC algorithms.

## 5.3 Quality Estimation

Before the final termination criterion is derived, this section defines a mechanism to estimate the quality of solutions found by the algorithm based on the regression results. In order to measure the solution quality, the approximation ratio of a solution $\varepsilon(\omega)$ to the optimal solution is introduced, which is defined in (5.13).

$$\varepsilon(\omega) = \frac{J(\omega) - J(\omega_{OPT})}{J(\omega_{OPT})} \tag{5.13}$$

Hereby $\omega_{OPT}$ denotes the optimal solution and thus $J(\omega) - J(\omega_{OPT})$ gives the quality distance between the current solution and the optimal solution. Usually, one has to compute $J(\omega_{OPT})$ to obtain the approximation ratio. By

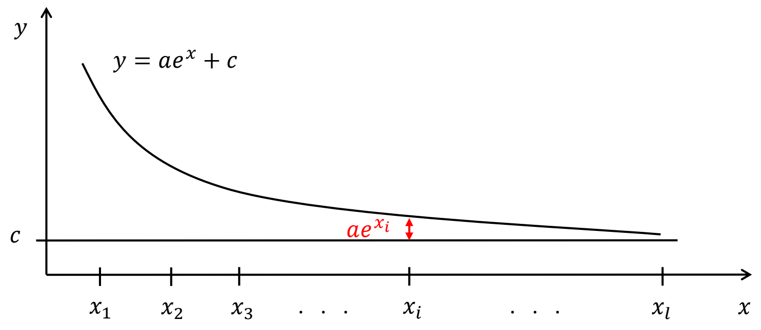


Figure 5.8: The exponential regression function

looking at the regression function shown in (5.2), the parameter $c$ (the convergence point of the function as shown in Figure 5.8) is in fact the value of the optimal solution, i.e., $J(\omega_{OPT}) = c$. Since the regression process shown in the previous sections presumes the absence of $c$, the parameters $a$ and $b$ can be estimated but $c$ still remains unknown. In other words, the exact value of $J(\omega_{OPT})$ is not available. As a result, the main challenge to obtain $\varepsilon(\omega)$ is to estimate the following two terms:

1. $J(\omega) - J(\omega_{OPT})$
2. $J(\omega_{OPT})$

In order to estimate $J(\omega) - J(\omega_{OPT})$, the regression function (5.2) is slightly reformulated. Thus the equation (5.14) is obtained.

$$ae^{bx_i} = y_i - c, \quad i = 1, ..., l \tag{5.14}$$

It is not hard to see that the term $ae^{bx_i}$ (cf. Figure 5.8) expresses the absolute difference between the value of the solution found until the $i$-th iteration and the optimal solution, which is in turn an estimator of $J(\omega) - J(\omega_{OPT})$. Thus the approximation ratio can be obtained in (5.15)

$$\varepsilon(\omega) = \frac{ae^{bx}}{J(\omega_{OPT})} \tag{5.15}$$

Now the only remaining obstacle is to compute $J(\omega_{OPT})$. Though the exact value of $J(\omega_{OPT})$ is not available, a lower bound of it (denoted by $J'(\omega_{OPT})$) is able to be computed by using Algorithm 5 and 6 for the RTSC and RTMC problems, respectively. Hereby, the main idea is to compute the optimal power consumption for each task and device independently, i.e., disregarding the schedulability test.

**Algorithm 5** Computation of $J'(\omega_{OPT})$ for RTSC

**Input:** $(\Gamma, \mathcal{C}, \mathcal{S}, \mathcal{R})$
**Output:** $J'(\omega_{OPT})$

```
1: $J'(\omega_{OPT}) \leftarrow 0$
2: **for all** task $\tau_i$ **do**
3:     $U(\tau_i) \leftarrow \frac{W(\tau_i) \cdot F(S_1)}{F(SC(\tau_i)) \cdot T(\tau_i)}$
4:     $J'(\omega_{OPT}) \leftarrow J'(\omega_{OPT}) + U(\tau_i) \cdot P(SC(\tau_i))$
5: **end for**
6: **for all** device $R_j$ **do**
7:     $U(R_j) = \sum_{R_j \in Dev(\tau_k)} U(\tau_k)$
8:     $J'(\omega_{OPT}) \leftarrow J'(\omega_{OPT}) + U(R_j) \cdot P(D_0^j)$
9: **end for**
```

| State | $F$ | $P$ (mW) | $T_{on \to off}$ | $T_{off \to on}$ | $P_{on \to off}$ | $P_{off \to on}$ | $T_{be}$ |
|---|---|---|---|---|---|---|---|
| $S_1$ | 1 | 800 | n.a | n.a | n.a | n.a | n.a |
| $S_2$ | 0.5 | 300 | n.a | n.a | n.a | n.a | n.a |
| $C_1$ | n.a | 50 | 2 ms | 2 ms | 50 mW | 50 mW | 4 ms |
| $D_0$ | n.a | 1000 | n.a | n.a | n.a | n.a | n.a |
| $D_1$ | n.a | 100 | 2 ms | 2 ms | 100 mW | 100 mW | 4 ms |

Table 5.1: The processor and device power model

For the RTSC problem, the critical speed for each task is first computed. By definition, this speed yields the optimal task AEC. Based on the critical speed, the device utilization can be derived. The total system energy consumption is a sum of energy consumption of all the tasks and devices according to their utilization. For the unused time, the power consumption is considered to be zero. Thus this algorithm produces a lower bound of the optimal solution.

For the RTMC problem, the basic idea is similar and the critical speed of all the tasks are calculated. However, in case of a multi-core platform where the processor cores have heterogeneous power models, the task critical speed can not be determined, because it is dependent on the processor core to which it is allocated and this will require a task partition. In this case, the task critical speed is computed for all the processor cores and the one with the least energy consumption is taken.

Though $J'(\omega_{OPT})$ is very optimistically estimated and not reachable in the reality, it is sufficient to substitute $J(\omega_{OPT})$ in (5.15) to obtain the approximation ratio.

One example instance of the RTSC problem is shown in Table 5.1 and 5.2. In

**Algorithm 6** Computation of $J'(\omega_{OPT})$ for RTMC

**Input:** $(\Gamma, \mathcal{O})$
**Output:** $J'(\omega_{OPT})$

```
1:  $J'(\omega_{OPT}) \leftarrow 0$
2:  **for all** task $\tau_i$ **do**
3:      $E \leftarrow \max$
4:      **for all** processor core $O_j$ **do**
5:          $U^j(\tau_i) \leftarrow \frac{W(\tau_i) \cdot F(S_1^j)}{F(SC^j(\tau_i)) \cdot T(\tau_i)}$
6:          $E^j \leftarrow U^j(\tau_i) \cdot P(SC^j(\tau_i))$
7:          **if** $E^j < E$ **then**
8:              $E \leftarrow E^j$
9:          **end if**
10:     **end for**
11:     $J'(\omega_{OPT}) \leftarrow J'(\omega_{OPT}) + E$
12: **end for**
```

| Task | $W$ (ms) | $T$ (ms) | $Dev$ | $SC$ |
|---|---|---|---|---|
| $\tau_1$ | 5 | 20 | $\varnothing$ | $S_2$ |
| $\tau_2$ | 10 | 40 | $\{R_1\}$ | $S_1$ |

Table 5.2: The task specification

this example, the critical speeds for $\tau_1$ and $\tau_2$ are $S_2$ and $S_1$, respectively. Thus the utilization of $\tau_1$ and $\tau_2$ (running at their corresponding critical speed) are 0.5 and 0.25, respectively. The device $R_1$ is only used by $\tau_2$ and therefore its utilization is 0.25 as well. Thus, $J'(\omega_{OPT})$ can be obtained.

$$J'(\omega_{OPT}) = 0.5 \cdot 800 \text{ mW} + 0.25 \cdot 800 \text{ mW} + 0.25 \cdot 1000 \text{ mW} = 850 \text{ mW} \tag{5.16}$$

Furthermore, Table 5.3 illustrates an example trace of running the HSASC

| $x$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| $J(\omega)$ (mW) | 1800 | 1300 | 1080 | 1080 | 980 | 920 | 900 | 900 |
| $\delta$ | - | 500 | 220 | 0 | 100 | 60 | 20 | 0 |
| $\delta'$ | - | 500 | 220 | 198 | 100 | 60 | 20 | 18 |
| $a$ | - | - | 2029 | 1902 | 1904 | 1912 | 2080 | 2021 |
| $b$ | - | - | -0.82 | -0.46 | -0.49 | -0.50 | -0.59 | -0.57 |
| $\varepsilon(\omega)$ | - | - | 0.20 | 0.35 | 0.19 | 0.11 | 0.04 | 0.02 |

Table 5.3: The data set indicating an execution trace of the HSASC algorithm

| State | $F$ | $P$ (mW) | $T_{on\rightarrow off}$ | $T_{off\rightarrow on}$ | $P_{on\rightarrow off}$ | $P_{off\rightarrow on}$ | $T_{be}$ |
|---|---|---|---|---|---|---|---|
| $S_1^1$ | 1 | 800 | n.a | n.a | n.a | n.a | n.a |
| $S_2^1$ | 0.5 | 300 | n.a | n.a | n.a | n.a | n.a |
| $C_1^1$ | n.a | 50 | 2 ms | 2 ms | 50 mW | 50 mW | 4 ms |
| $S_1^2$ | 1 | 800 | n.a | n.a | n.a | n.a | n.a |
| $S_2^2$ | 0.5 | 300 | n.a | n.a | n.a | n.a | n.a |
| $C_1^2$ | n.a | 50 | 2 ms | 2 ms | 50 mW | 50 mW | 4 ms |

Table 5.4: The power model of a dual-core processor

| Task | $W$ (ms) | $T$ (ms) | $Dev$ | $SC^1$ | $SC^2$ |
|---|---|---|---|---|---|
| $\tau_1$ | 10 | 20 | $\varnothing$ | $S_2^1$ | $S_2^2$ |
| $\tau_2$ | 15 | 40 | $\varnothing$ | $S_2^1$ | $S_2^2$ |

Table 5.5: The task specification

algorithm for 8 iterations (shown in the first row). One assumes that the solution values (shown in the second row) are artificially generated (by simulating an exponentially decreasing trend). The actual value improvement at each iteration can then be computed and is shown in the third row. The fourth row shows the smoothed data. Hereby for the sake of simplicity, the smoothing parameter $\alpha$ is assumed to be constant and equal to 0.01. Based on (5.10) and (5.9), the regression parameters $a$ and $b$ are estimated for each iteration. Finally, according to (5.15) the approximation ratio for each iteration can be computed (shown in the last row).

For the RTMC problems, one example instance is shown in Table 5.4 and 5.5. Both processor cores are in one cluster and thus share the same power model. Note that the critical speed of $\tau_1$ and $\tau_2$ on both cores are listed in Table 5.5. The utilization of both tasks running at the critical speed then can be easily computed, which are 1 and 0.75, respectively. Thus, $J'(\omega_{OPT})$ can be obtained.

| $x$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| $J(\omega)$ (mW) | 3400 | 2400 | 1900 | 1900 | 1700 | 1600 | 1520 | 1500 |
| $\delta$ | - | 1000 | 500 | 0 | 200 | 100 | 80 | 20 |
| $\delta'$ | - | 1000 | 500 | 450 | 200 | 100 | 80 | 20 |
| $a$ | - | - | 4000 | 4106 | 4060 | 4191 | 4076 | 4548 |
| $b$ | - | - | -0.69 | -0.40 | -0.49 | -0.55 | -0.52 | -0.60 |
| $\varepsilon(\omega)$ | - | - | 0.36 | 0.59 | 0.25 | 0.11 | 0.08 | 0.03 |

Table 5.6: The data set indicating an execution trace of the HSAMC algorithm

$$J'(\omega_{OPT}) = 1 \cdot 800 \text{ mW} + 0.75 \cdot 800 \text{ mW} = 1400 \text{ mW} \tag{5.17}$$

Furthermore, Table 5.6 shows an example trace of running the HSAMC algorithm for 8 iterations (shown in the first row). Hereby the computation of various parameters is similar to the case with the RTSC problem shown before.

**Algorithm 7** Heuristic Search Algorithm for RTSC (HSASC)

**Input:** The solution space $\Omega = \{assign_1, assign_2, ..., assign_{|\mathcal{S}|^n}\}$
**Output:** A solution $assign_{best}$

```
1: Generate $assign_{init}$ with $\forall \tau \in \Gamma : assign_{init}(\tau) = S_1$
2: $assign_{best} \leftarrow assign_{init}$, $assign_{curr} \leftarrow assign_{init}$, $i \leftarrow 1$
3: $\sum X \leftarrow 0$, $\sum X^2 \leftarrow 0$, $\sum Y \leftarrow 0$, $\sum XY \leftarrow 0$, $\delta \leftarrow 0.01$
4: while C1 and C2 are not met do
5:     Generate $assign'$ from $assign_{curr}$ and a random number $r \in [0,1]$
6:     if $r \leq Pr_{accept}(assign_{curr}, assign', i)$ then
7:         $assign_{curr} \leftarrow assign'$
8:         if $J(assign_{curr}) < J(assign_{best})$ then
9:             $\delta \leftarrow J(assign_{best}) - J(assign_{curr})$, $assign_{best} \leftarrow assign_{curr}$
10:        else
11:            Compute $\alpha$ based on (5.12), $\delta \leftarrow (1-\alpha) \cdot \delta$
12:        end if
13:    end if
14:    $\sum X \leftarrow \sum X + i$, $\sum X^2 \leftarrow \sum X^2 + i^2$
15:    $\sum Y \leftarrow \sum Y + ln(\delta)$, $\sum XY \leftarrow \sum XY + i \cdot ln(\delta)$
16:    Compute $a$ and $b$ according to (5.10) and (5.9), $i \leftarrow i + 1$
17: end while
```

## 5.4 Termination Criterion

The termination criterion is mainly derived based on the approximation ratio. More specifically, the HSASC and HSAMC algorithms terminate, if any of the two following conditions is satisfied.

- C1: $\varepsilon(\omega) < \beta$, $a > 0$ and $b < 0$ hold, where $\omega$ is the currently produced solution.
- C2: The algorithm reaches a predefined threshold $I_{th}$ of the iteration number.

$\beta$ and $I_{th}$ are the constants to be decided by the user. The condition C1 considers the approximation ratio $\varepsilon(\omega)$, which gives the information of how close a solution found by the heuristic search algorithm is to the optimal solution. If the distance is smaller than the one required by the user, then the algorithm terminates. The second condition C2, on the one hand, ensures that algorithms will definitely terminate and on the other hand, gives the user a possibility to define an upper bound of the affordable run-time.

## 5.5 Chapter Summary

**Algorithm 8** Heuristic Search Algorithm for RTMC (HSAMC)

**Input:** The solution space $\Omega_{RTMC}$
**Output:** A solution $(alloc, assign)_{best}$

```
1: Generate $(alloc, assign)_{init}$
2: $(alloc, assign)_{best} \leftarrow (alloc, assign)_{init}$
3: $(alloc, assign)_{curr} \leftarrow (alloc, assign)_{init}$
4: $i \leftarrow 1$
5: $\sum X \leftarrow 0$, $\sum X^2 \leftarrow 0$, $\sum Y \leftarrow 0$, $\sum XY \leftarrow 0$, $\delta \leftarrow 0.01$
6: while C1 and C2 are not met do
7:     Generate $(alloc, assign)'$ from $(alloc, assign)_{curr}$
8:     Generate a random number $r \in [0,1]$
9:     if $r \leq Pr_{accept}((alloc, assign)_{curr}, (alloc, assign)', i)$ then
10:        $(alloc, assign)_{curr} \leftarrow (alloc, assign)'$
11:        if $J((alloc, assign)_{curr}) < J((alloc, assign)_{best})$ then
12:            $\delta \leftarrow J((alloc, assign)_{best}) - J((alloc, assign)_{curr})$
13:            $(alloc, assign)_{best} \leftarrow (alloc, assign)_{curr}$
14:        else
15:            Compute $\alpha$ based on (5.12), $\delta \leftarrow (1-\alpha) \cdot \delta$
16:        end if
17:    end if
18:    $\sum X \leftarrow \sum X + i$, $\sum X^2 \leftarrow \sum X^2 + i^2$
19:    $\sum Y \leftarrow \sum Y + ln(\delta)$, $\sum XY \leftarrow \sum XY + i \cdot ln(\delta)$
20:    Compute $a$ and $b$ according to (5.10) and (5.9), $i \leftarrow i+1$
21: end while
```

This chapter mainly provides a run-time behavior analysis of the HSASC and HSAMC algorithms based on the regression technique. Because both algorithms are following simulated annealing heuristic, there is neither guarantee of finding a globally optimal solution nor information about the solution quality. Some related work are first reviewed in terms of convergence theory, however, their results are either only relevant in theory or only addressing a specific optimization problem in the complexity class $\mathcal{P}$. There is no direct answer for the questions Q1 and Q2 that are raised at the beginning of this chapter. Therefore, this chapter proposes to apply the regression technique at run-time to simulate and thus be able to predict the behavior of the algorithms. Based on the regression result, a mechanism is further proposed to estimate the quality of solutions and finally derive a termination criterion of the algorithms.

As a conclusion of this chapter, Algorithm 7 and 8 give a complete overview of the HSASC and HSAMC algorithms with integration of the proposed termination criterion, respectively.

# Chapter 6

# ES-AS: An Online Approach

Naturally, the HSASC and HSAMC algorithms are often implemented as offline programs to compute the solution. However, there are several problems arising in this context. One of the most difficult obstacles is to evaluate the solution value $J(\omega)$. The main contribution of this chapter is to address all these issues and propose an approach, ES-AS, to overcome them. Briefly, the ES-AS approach runs the HSASC and HSAMC algorithms in a fully online and adaptive fashion. The main part of this chapter was published in [HM12c] and [HM13b].

## 6.1 Motivation

Both HSASC and HSAMC algorithms are simulated annealing based meta-heuristics. As a nature, these algorithms are often implemented in an offline fashion. One of the most important requirements in meta-heuristics is the capability to efficiently evaluate the solution value. In many combinatorial optimization problems, it can be expressed in a closed form and therefore presents no real challenge. In the HSASC and HSAMC algorithms, however, evaluation of the solution value is not a trivial job. As shown in Chapter 4, the solution value is defined as the system energy consumption over a hyper period. The toughest part is to analyze the energy consumed by components when they are idle. The main reason is to be accounted for the consideration of non-negligible DPM state switching overhead. Depending upon the length of an idle interval, different calculation formulas should be used. More specifically, if the idle interval is larger than the corresponding break-even time, then the power consumption of a low power state is to be used. On the contrary, the component is active and thus the power consumption of the active state is to be applied. As a result, in order to evaluate a solution, all the idle intervals must be analyzed. Obviously, this is a very time consuming job, because an offline analysis of idle intervals requires exact knowledge of the start, preemption, resume, and completion time of all the tasks. This effort

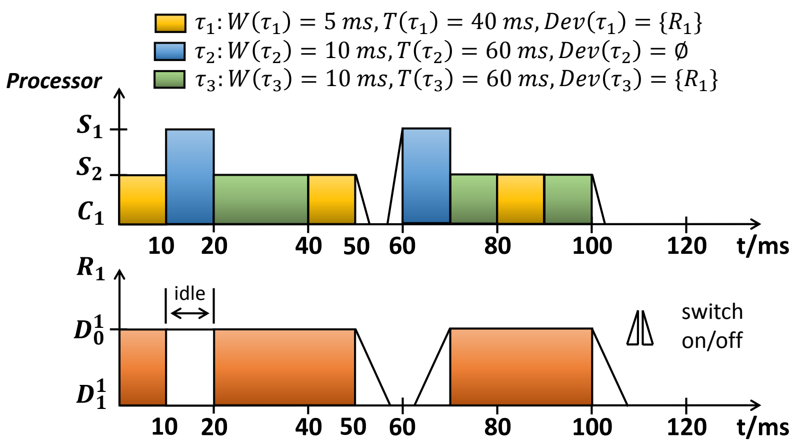


Figure 6.1: An example with a single-core processor and a device

at least has the same computation complexity as the task worst case response time analysis, which is pseudo-polynomial [But11].

In order to overcome this problem, this chapter introduces the ES-AS approach, which runs the algorithms in a fully online fashion. As will be explained later, the evaluation of solutions then becomes a easy work. The basic concept of the proposed approach is mainly inspired by the general idea of Organic Computing [Sch05], which describes a technical system that is able to autonomically adapts itself to environment changes. The fundamental characteristics of such systems are referred to as the so-called self-X properties including self-configuration, self-healing, self-optimization, etc.. This work follows this basic principle to make systems self-adaptive in terms of energy optimization. In other words, an online approach provides the advantage that system run-time and dynamic information can be taken into consideration, whereas offline approaches lack this ability. In the context of real-time systems, the dynamic information is twofold. The first type is related to dynamic slack, which describes the unused time of a task due to its earlier completion than the WCET. The second type of information concerns system changes, such as a new task or device joining or leaving the system at run-time. The goal is to react on this information and even make use of it for further energy reduction if possible.

In addition, the proposed online approach offers a positive side effect as well, which is particularly related to the RTSC problem. As shown in the previous chapters, the HSASC algorithm produces a solution assigning each task with an operating speed. However, there is no information about power state scheduling (cf. Problem 3.4.1), which decides, for instance, when a device can be switched off and when it should be switched on. By definition of break-even time, the device can not be blindly turned off whenever it becomes idle. Instead, a shutdown is dependent on the length of the upcoming idle interval. Besides, the device needs to be switched on a little ahead of

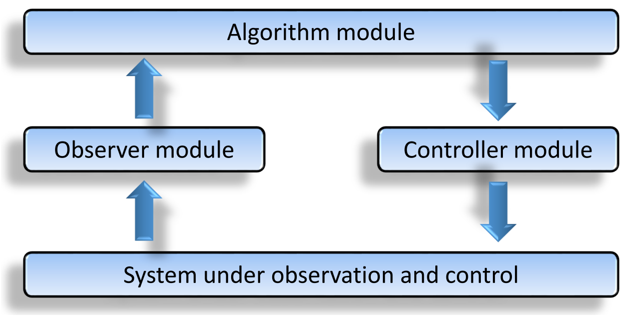


Figure 6.2: The observer/controller architecture of organic computing

the dispatching time of the next requiring task, because otherwise the task is delayed and its deadline may be jeopardized. In a preemptive real-time system, analyzing or predicting the length of idle intervals for such devices at run-time is not computationally feasible [DA08a]. That is, deriving a power state scheduling at run-time is in general not possible. The ES-AS approach, however, tries to overcome this obstacle by using a special event recording activity.

To emphasis the importance of power state scheduling, a simple example is illustrated in Figure 6.1, where three real-time tasks are running under the EDF scheduling algorithm. The single-core processor supports two P-states with $F(S_1) = 1$ and $F(S_2) = 0.5$. $\tau_1$ and $\tau_3$ are assigned with $S_2$ and therefore their actual run-time are twice the original WCET. $\tau_2$ is assigned with $S_1$ and its run-time is equal to $W(\tau_2)$. Furthermore, according to the task specification shown in Figure 6.1, the device $R_1$ is required by the tasks $\tau_1$ and $\tau_3$. The power state scheduling has the job, for instance, to decide at the time point 10 ms whether $R_1$ should be switched off. In this example, the marked idle time is too short to shut down the device and thus it has to remain in the active state. On the contrary, at 50 ms the device becomes idle again and is switched off to a low power state, as the idle time is long enough. Afterward, the device $R_1$ is activated before the actual required time, so that $\tau_3$ is not delayed.

In what follows, an overview of the ES-AS approach is given first and the details of its application on the RTSC and RTMC problems are explained subsequently.

## 6.2 Overview of ES-AS Approach

Following the basic idea behind Organic Computing, the goal is to find a way that the HSASC and HSAMC algorithms can be implemented while being able to be adaptive to the dynamically changing environment. Once a system is set up, it runs completely autonomically. Whenever a dynamic change occurs, the system is supposed to be reactive and takes the change into consideration by computing a new solution that is more suitable. Hereby, a generic observer/controller architecture [Ric+06] is adopted and its simplified version is illustrated in Figure 6.2.

In order to make the original system self-adaptive, several additional modules need to be implemented. The observer module is responsible to monitor system behavior and forward the observation to the algorithm module if any changes happen. In case of a dynamic change, the algorithm module then takes the responsibility to adapt the current solution to the change. Finally, the controller module applies the new solution back to the system.

In the concrete context of RTSC and RTMC, the observer module should be able to detect the above mentioned two types of dynamic information:

- dynamic slack
- if there is any task or device joining or leaving the system

The treatment for dynamic slack is straightforward, since the unused time can be further exploited for more energy saving. More details will be explained later in the subsequent sections. The second type of information, however, presents a bigger problem, because a new solution might need to be computed.

A naive answer might be executing the entire algorithm, i.e., HSASC or HSAMC, to compute the new solution before the actual real-time tasks start, once system changes are detected. However, it is not suitable for hard real-time systems due to the non-deterministic characteristic of meta-heuristics, i.e., the algorithm may take too long and result in a deadline miss, because the actual tasks are unjustifiably procrastinated. Hereby the main challenge is to integrate the HSASC and HSAMC algorithms into the running real-time systems while meeting all the deadlines. The basic idea is to take the advantage of the common feature of both domains, namely ”iterative”. The algorithms are iterative, because they iteratively improve candidate solutions, while real-time systems with periodic tasks are also iterative in terms of the repeated task execution in each hyper period. In the approach, each iteration of the algorithms is mapped to a hyper period. In other words, in each hyper period a candidate solution is explored and evaluated. This stage is called Exploration Stage (ES) and is stopped when the termination criterion is fulfilled. For the remaining time it is called Application Stage (AS), because in this stage the best solution found in the exploration stage is applied. This is also the reason that the approach takes the name ES-AS. In the next sections, the details of the ES-AS approach are presented for the RTSC and RTMC problems, respectively.

## 6.3 ES-AS Approach for RTSC

This section describes the ES-AS approach applied on the RTSC problem, where the system-wide energy consumption is to be optimized. Before ES and AS are explained in detail, several assumptions are made. First of all, all the tasks are assumed to run until their WCET during ES. If a task, nevertheless, completes earlier than its WCET, then its execution time is artificially prolonged to the WCET. The tasks, however, are allowed to run shorter than their WCET in AS. In other words, ES only exploits static slack and dynamic slack will be considered during AS. Moreover, since a system has no knowledge of power state scheduling at the beginning, all the devices plus the processor are kept in the active state even they are idle. In this manner, no task will be delayed due to DPM state switching overhead. However, a power state scheduling will be derived during ES and applied in AS.

### 6.3.1 Exploration Stage

In this stage, the main goal is trying to find the best solution. Mainly one candidate solution is explored in one hyper period. At run-time two main activities are taken to achieve this goal.

**Algorithm 9** Algorithm Activity at the end of each hyper period in ES (RTSC)

```
1: Get $J(assign_{curr})$
2: Compare $J(assign_{curr})$ and $J(assign_{prev})$
3: **if** $assign_{curr}$ is not accepted **then**
4:     $assign_{curr} \leftarrow assign_{prev}$
5: **end if**
6: Estimate the regression parameters $a$ and $b$
7: **if** Termination criterion not met **then**
8:     Generate $assign_{new} \in N(assign_{curr})$
9:     $assign_{prev} \leftarrow assign_{curr}$
10:    **if** $assign_{new}$ is schedulable **then**
11:        $assign_{curr} \leftarrow assign_{new}$
12:    **end if**
13: **else**
14:    Terminate ES and enter AS
15: **end if**
```

The first activity with the name Algorithm Activity (AA) takes place at the end of each hyper period. AA mainly performs the work specified for each iteration in the HSASC algorithm. Algorithm 9 shows a pseudo implementation of this activity. The value of the current solution $assign_{curr}$ has to be first evaluated. Since the system energy is recorded at run-time, which will be explained later, the solution value can be easily obtained. Afterward the acquired value is to be compared with the value of the solution from the previous hyper period $assign_{prev}$. According to acceptance probability a movement to the current solution is made. Based on the new value, the regression process is carried out to estimate the regression parameters and thus the approximation ratio. If the termination criterion is not yet met, the next solution $assign_{new}$ will be generated, which is a neighbor solution of the current solution. Before the next hyper period starts, AA decides whether the new solution will make the system schedulable. If it is feasible, then the new solution is used in the next hyper period, otherwise the current solution is used.

The second activity with the name Recording Activity (RA) happens at each scheduling point. As mentioned in Chapter 3, the term "component" in this chapter is also used in a general sense and referred to as either a device or a processor. Mainly, RA records two types of data.

1. Activation and deactivation events of all the components
2. System energy consumption.

The events are collected over each hyper period in an event list per component. Each event contains two kinds of information: i) the time stamp when it is recorded and ii) if it is an activation event or deactivation event. Figure 6.3 shows an example illustrating the events $e_1$, $e_2$, $e_3$, and $e_4$ recorded into the event list of the device $R_1$ and $e_5$, $e_6$, $e_7$, $e_8$, $e_9$, and $e_{10}$ into the event list of the processor. All the events of $R_1$ are related to the execution of the task $\tau_1$, because $R_1 \in Dev(\tau_1)$. Figure 6.3 shows also that $R_1$ is kept always active as mentioned earlier, even when it is not needed (between 40 ms and 80 ms). The recorded events intuitively reflect the component behavior, when it should be activated and when it could be deactivated. These events will be used in AS to derive the power state scheduling. Another observation is that the event list of the processor actually collects the activation and deactivation events of all the tasks, i.e., the task schedule.

The second type of data to be recorded in RA is the system energy recording, which is quite straightforward. Since RA takes place at run-time, it is obvious when a task starts and finishes. At each scheduling point RA records the energy consumption of involved components for the time interval between the previous recorded event and the current event, then this is cumulatively added to the total energy consumption for the current hyper period. At the end of a hyper period the total system energy consumption is obtained without any extra effort, which is exactly the solution value. In Figure 6.3, for instance, at the scheduling point 40 ms the task $\tau_1$ completes. The involved

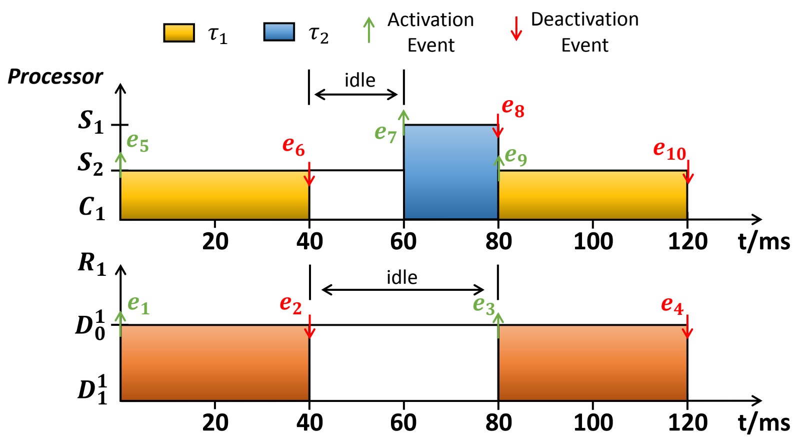


Figure 6.3: Events recording in one example with $Dev(\tau_1) = \{R_1\}$ and $Dev(\tau_2) = \varnothing$

components are the processor and the device $R_1$. Here the discussion is focused on the computation of energy consumption for $R_1$. When the system runs to the time point 40 ms, where the event $e_2$ is stored, RA records the energy consumption of $R_1$ for the first time. The involved interval is from the previous event $e_1$ to the current event $e_2$. In this interval $R_1$ was active and therefore the energy consumption is computed by means of

$$E = P(D_0^1) \cdot (e_2.t - e_1.t) \tag{6.1}$$

provided that $e_i.t$ denotes the time stamp of $e_i$. As the task execution continues, $R_1$ becomes idle and the idle time lasts from $e_2$ to $e_3$. At the time point 80 ms, where $e_3$ is recorded, RA computes the idle length $l = e_3.t - e_2.t$ and afterward the energy consumption $E$ by (6.2) for this idle interval. Note that the scheduling point at 60 ms is not relevant for $R_1$, because it is not involved at that point.

$$E = \begin{cases} E_{on\to off} + E_{off} + E_{off\to on} & \text{if } l > T_{be}(D_1^1) \\ P(D_0^1) \cdot l, & \text{otherwise} \end{cases} \tag{6.2}$$

Hereby $E_{on\to off}$ is the energy consumption for switching off $R_1$, $E_{off\to on}$ is the energy consumption for switching on $R_1$ and $E_{off}$ is the energy consumed by $R_1$ when it is in the low power state $D_1^1$. In general, if a component supports multiple low power states, the state with the largest break-even time that is less than the length of the idle interval is selected. This example assumes that $R_1$ supports only one low power state. $E_{on\to off}$, $E_{off\to on}$ and $E_{off}$ can be expressed by (6.3). Note that the energy recording for each component needs only $O(1)$ operation at each scheduling point.

$$\begin{aligned} E_{off} &= P(D_1^1) \cdot (l - T_{on\rightarrow off}(D_1^1) - T_{off\rightarrow on}(D_1^1)) \\ E_{on\rightarrow off} &= P_{on\rightarrow off}(D_1^1) \cdot T_{on\rightarrow off}(D_1^1) \\ E_{off\rightarrow on} &= P_{off\rightarrow on}(D_1^1) \cdot T_{off\rightarrow on}(D_1^1) \end{aligned} \tag{6.3}$$

As a short summary, the recorded events reflect the time behavior of components, when they should be activated and when they could be deactivated. The recorded energy is the energy consumed by the components, if they are switched on and off according to the recorded events. A pseudo implementation of RA is shown in Algorithm 10.

**Algorithm 10** Recording Activity at each scheduling point in ES (RTSC)

```
 1: if There is a task $\tau_i$ finishing or being preempted then
 2:     Add a deactivation event into the list of the processor
 3:     Record the energy consumed by the processor in the last interval (de-
        pending on the used P-state)
 4:     for all $R_j \in Dev(\tau_i)$ do
 5:         Add a deactivation event into the list of $R_j$
 6:         Record the energy consumed by $R_j$ in the last interval based on (6.1)
 7:     end for
 8: end if
 9: if There is a task $\tau_i$ starting or being resumed then
10:     Add an activation event into the list of the processor
11:     Record the energy consumed by the processor in the last interval
12:     for all $R_j \in Dev(\tau_i)$ do
13:         Add an activation event into the list of $R_j$
14:         Record the energy consumed by $R_j$ in the last interval based on (6.2)
15:     end for
16: end if
```

### 6.3.2 Application Stage

In this stage the best solution found in ES is applied and a power state scheduling based on the recorded events is derived and used. By definition, a power state scheduling has the work to decide when a component is to be switched off and when to be switched on. For this purpose an activity called DPM Activity (DA) is taken at each scheduling point. If there is a task finishing or being preempted at a scheduling point, then the components required by the task can be potentially switched off to a low power state, however, this decision is dependent on the length of the upcoming idle interval. Other existing online approaches are often applying complicated techniques for estimating the next activation time (even with task procrastination), which are usually very time-consuming. On the contrary, DA can simply consult the

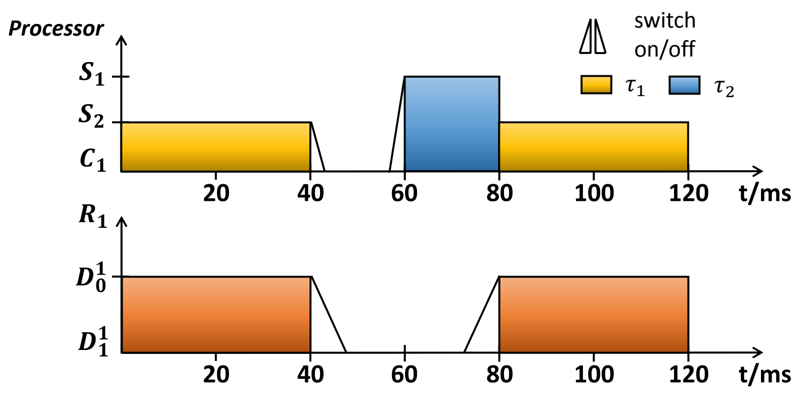


Figure 6.4: Application stage in one example with $Dev(\tau_1) = \{R_1\}$ and $Dev(\tau_2) = \varnothing$

event list to compute the length of the next idle interval, which is much more efficient. According to this information the decision whether a component has to be switched off to a particular low power state is made. If a component is switched off to a low power state, it also needs to be switched on a little ahead of the next required time. If the idle time is shorter than the break-even time of all the low power states (i.e., the component can not be shut down at all), the component stays in the active state. Figure 6.4 illustrates this DPM activity without consideration of dynamic slack. This example assumes that the solution and event lists are obtained from Figure 6.3. Since task run-time variation is not considered, the task execution behaves exactly the same as in Figure 6.3. At the scheduling point 40 ms, the task $\tau_1$ completes and the processor and $R_1$ can be potentially put into a low power state. After consulting the event list of the processor and $R_1$, respectively, the processor will be required by 60 ms and $R_1$ will be required by 80 ms. For example, both idle intervals are large enough, then both processor and $R_1$ are switched to a proper low power state. Note that the components are switched on a little ahead of their actual required time. The switching on and off processes are illustrated by triangles in Figure 6.4.

As mentioned before, dynamic slack can be explored in AS. Here the dynamic slack reclaiming mechanism is adopted. More specifically, the unused execution time of a task is utilized for additional power saving through the DPM technique. Mainly the components can be switched off earlier than the worst case. Figure 6.5 shows the same example as in Figure 6.4, but all the tasks are finishing earlier than their WCET. At the scheduling point where the task $\tau_1$ finishes (at 30 ms), DA can consult the event list of the processor and $R_1$, respectively. As a consequence, it is obvious that the processor will be required at 60 ms and $R_1$ will be required at 80 ms. This information will guide the DPM activity to compute the length of the upcoming idle interval, which can be used to make the decision whether a component should be switched off to a low power state. One important note here is that all the tasks

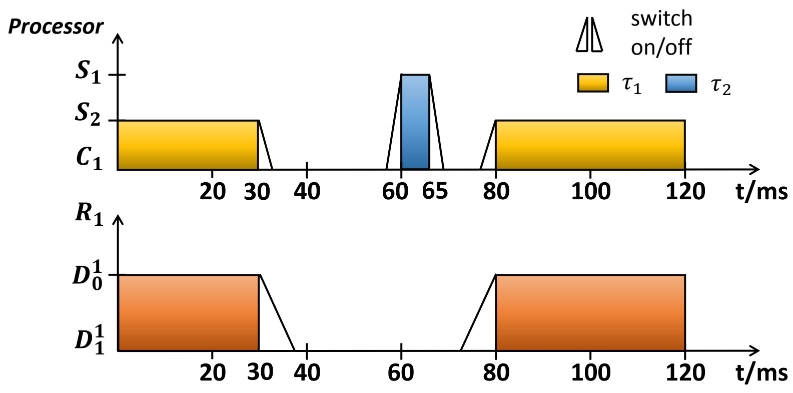


Figure 6.5: Dynamic slack reclaiming in one example with $Dev(\tau_1) = \{R_1\}$ and $Dev(\tau_2) = \varnothing$

are always activated at the recorded activation time. If a task is ready earlier than the recorded time due to the earlier completion of previous tasks, then it needs to be delayed to the recorded activation time, because only at that time one can guarantee that all the required components are in the active state. The second instance of the task $\tau_1$ in Figure 6.5 illustrates this situation. It becomes ready after the task $\tau_2$ completes (at 65 ms), however, it should wait until its recorded activation time, which is at 80 ms. A pseudo implementation of the DPM activity at each scheduling point is shown in Algorithm 11. The variable *index* denotes the current index in the current event list and is initialized with 1.

Generally, the ES-AS approach is launched at system start and runs from ES to AS. As soon as there are any system changes, such as a task or device joining or leaving the system, however, only allowed at hyper period boundaries, the approach will start over with the calculation of the new initial solution and run from ES to AS again. Figure 6.6 shows the activities taken in the ES-AS approach at a glance.

## 6.3.3 Correctness and Complexity

This section shows the correctness and complexity of the ES-AS approach. The correctness ensures the system schedulability in terms of meeting the task deadlines. The complexity gives the run-time overhead analysis of the approach.

**Theorem 6.3.1.** *The ES-AS approach always guarantees system schedulability.*

*Proof.* Since the ES-AS approach is divided into two stages, the system schedulability is proven for ES and AS, respectively. In ES one solution is explored in one hyper period. Since the initial solution obtained by assigning

**Algorithm 11** DPM Activity at each scheduling point in AS (RTSC)

```
1: if There is a task $\tau_i$ finishing or being preempted then
2:     for all $R_j \in Dev(\tau_i)$ and the processor do
3:         Retrieve the next activation event according to index and compute
           the length of the next idle interval. Based on the length it is to be
           decided, whether the component can be switched off to a proper low
           power state. If the component is switched off, it will be waked up a
           little ahead of the next activation time.
4:         Advance the variable index in the event list of $R_j$ and the processor.
5:     end for
6: end if
7: if There is a task $\tau_i$ starting or being resumed then
8:     if Current time is equal to the recorded time then
9:         Run the task $\tau_i$
10:        for all $R_j \in Dev(\tau_i)$ and the processor do
11:            Advance the variable index in the event list of $R_j$ and the proces-
               sor
12:        end for
13:    else
14:        Wait until the recorded time
15:    end if
16: end if
```

all the tasks with the maximal speed is clearly schedulable (otherwise there is no valid solution at all), in the first hyper period there will be no task missing its deadline. Note that in ES all the components are never switched off (even when they are idle), thus no task will be delayed due to state switching overhead. This is also the reason why the traditional schedulability test via utilization can be performed. At the end of the first hyper period the solution for the next hyper period is generated. As a schedulability test is carried out, and only if the test is positive, the generated solution will be used in the next hyper period (otherwise the initial solution is used), thus the schedulability is guaranteed as well. The procedure repeats at the end of each hyper period, therefore the solution used in each hyper period is obviously feasible.

In AS the best solution found in ES is applied and obviously this solution is schedulable. Furthermore, since all the components are switched on as recorded and all the tasks are activated as recorded, no task will be delayed and therefore no task will finish later than its recorded finishing time, which is obtained under worst case condition. Therefore no task will miss its deadline. In total, the system schedulability is always guaranteed. □

**Theorem 6.3.2.** *The ES-AS approach has a complexity of $O(m \cdot c)$ at each scheduling point and $O(n)$ at each hyper period boundary. $n$ is the number*

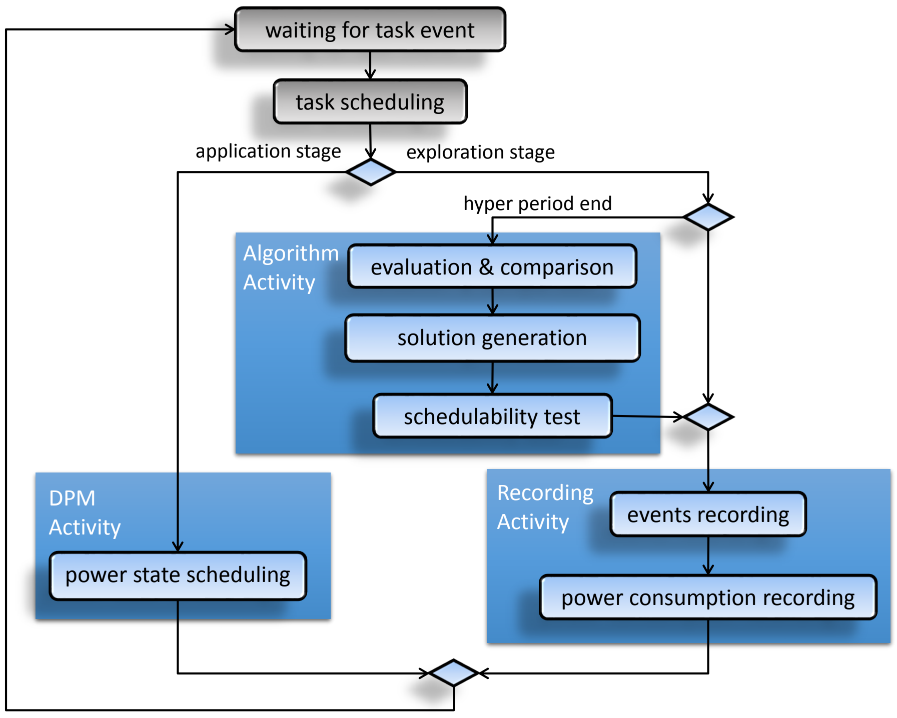


Figure 6.6: Overview of the activities taken in ES-AS approach

*of tasks, m is the number of devices and c is the maximal number of supported low power states by a component.*

*Proof.* The theorem is proven in two steps. In the first step the complexity at each hyper period boundary is shown and the complexity at each scheduling point is proven in the second step.

Looking at Figure 6.6, the only activity taken at hyper period boundaries is AA. At the end of each hyper period, it starts with the evaluation of the current solution. The evaluation takes $O(1)$, because through run-time energy recording the energy consumption of a solution is automatically available at the end of a hyper period. Afterward, the current solution is compared with the previous solution to decide whether a movement to the current solution should be made. This clearly takes $O(1)$ time. Depending on the acceptance result, a neighbor solution needs to be generated for the next hyper period. Though only the speed assignment of one task needs to be changed, generating the random number based on a non-uniform distribution has linear complexity $O(n)$. As shown in Chapter 4, the non-uniform distribution of the task selection probability is caused by the penalty function. After the generation process, the new solution is to be tested whether it can ensure system schedulability. This is usually performed by means of a processor utilization test. For both EDF and RM scheduling algorithms, the test takes $O(1)$, because only the utilization of the changed task needs to be updated. As a summary, AA takes $O(n)$ time in total and thus the time complexity of the ES-AS approach at hyper period boundaries is $O(n)$.

There are two activities taken at each scheduling point. The first one is RA, which mainly records events and energy consumption. Obviously, the recording process for a single component takes only $O(1)$ . At one scheduling point there can be at most $m$ components getting involved, where $m$ is the number of components in the system including all the devices and the processor. Therefore RA takes $O(m)$ time at each scheduling point. The second activity is DA, which takes care of the job of the power state scheduling. As shown in Algorithm 11, the most time consuming work is described at the line 3. Hereby DA needs to investigate the break-even time of all the low power states and decide whether the component can be switched off to a particular low power state. If the maximal number of the available low power states for a component is assumed to be $c$, then the time complexity of DA is $O(c)$ for each component. In summary, the run-time overhead of the ES-AS approach at each scheduling point is $O(m \cdot c)$ in total. □

Since $m$ and $c$ are often relatively small and constant, the run-time overhead of ES-AS approach at each scheduling point can be considered as $O(1)$.

## 6.4 ES-AS Approach for RTMC

This section describes the ES-AS approach applied on the RTMC problem, where the energy consumption of a set of real-time tasks running on a cluster-based multi-core processor is to be optimized. In general, it is quite similar to the application on the RTSC problem. The approach is also divided into two stages: ES and AS. The three main activities, AA, RA and DA, are taken as well. However, there are some difference in their implementations.

### 6.4.1 Exploration Stage

The main goal in ES is to find the best solution using the HSAMC algorithm. Hereby ES contains all three activities, the Algorithm Activity, the Recording Activity and the DPM activity. Note that this is different to the case of the RTSC problem, where ES contains only the first two activities. Moreover, ES here assumes that all the tasks run until their WCET. The same assumption is made as well in case of the RTSC problem.

AA behaves the same as for the RTSC problem, except that the work specified in the HSAMC algorithm is taken, instead of the HSASC algorithm. More specifically, the current solution is first evaluated and compared with the previous solution. Based on acceptance probability a decision is made, whether the current solution is accepted or not. Afterward a neighbor solution is generated for the upcoming hyper period. In order to ensure system schedulability, if the generated solution is not schedulable, the current solution is used in the upcoming hyper period. In other words, a non-schedulable solution will never be selected and used during online execution. Algorithm 12 shows the pseudo implementation of the activity. Hereby task migration overhead is not explicitly considered, because it occurs only at hyper period boundaries. However, if a task migration indeed takes significant time, it can be specified as a separate task running always at the maximal frequency and taken into account in the schedulability analysis.

**Algorithm 12** Algorithm Activity at each hyper period boundary in ES (RTMC)

```
 1: Get $J((alloc,assign)_{curr})$ and compare it with $J((alloc,assign)_{prev})$
 2: if $(alloc,assign)_{curr}$ is not accepted then
 3:     $(alloc,assign)_{curr} \leftarrow (alloc,assign)_{prev}$
 4: end if
 5: Estimate the regression parameters $a$ and $b$
 6: if Termination criterion not met then
 7:     Generate $(alloc,assign)_{new} \in N((alloc,assign)_{curr})$
 8:     $(alloc,assign)_{prev} \leftarrow (alloc,assign)_{curr}$
 9:     if $(alloc,assign)_{new}$ is schedulable then
10:         $(alloc,assign)_{curr} \leftarrow (alloc,assign)_{new}$
11:     end if
12: else
13:     Terminate ES and enter AS
14: end if
```

The second activity is RA, which takes place at each scheduling point and mainly records two types of matters, energy consumption and heuristic information. The energy consumption recording is quite similar to the case of the RTSC problem and is shown in Algorithm 13. However, the heuristic information recording is new and in fact replaces the event recording in the case of the RTSC problem. The explanation will be found later.

In Algorithm 13, the parameters $t_{last}$ and $t_{current}$ store the time stamps of the last and current scheduling point, respectively. The parameter $ls$ is the state used by the processor during the time interval $[t_{last}, t_{current}]$. Note that $ls$ could be a P-state or a low power state. The variable $e$ stores the cumulatively recorded system energy consumption so far.

The heuristic information update is basically for the computation of task penalty, which is required at each hyper period boundary to generate a neighbor solution. As shown in Chapter 4, the task penalty function is defined in (4.16) and mainly comprises two parts. The first part is related to the critical

**Algorithm 13** Energy Recording at each scheduling point in ES (RTMC)

```
1: if $ls$ is a P-state then
2:     $temp \leftarrow P(ls) \cdot (t_{current} - t_{last})$
3: else if $ls$ is a C-state then
4:     $temp \leftarrow P_{on \to off}(ls) \cdot T_{on \to off}(ls) + P_{off \to on}(ls) \cdot T_{off \to on}(ls) + P(ls) \cdot (t_{current} - t_{last} - T_{on \to off}(ls) - T_{off \to on}(ls))$
5: end if
6: $e \leftarrow e + temp$
7: $t_{last} \leftarrow t_{current}$
```

speed and can be easily calculated. However, the second part is not trivial, because it needs the information about unbalanced task execution during the last hyper period. The ES-AS approach solves this problem by updating task penalty values cumulatively at each scheduling point. The basic principle is analogue to the power consumption recording. Thus the heuristic information update process is part of the recording activity. Algorithm 14 shows the heuristic information update process.

**Algorithm 14** Heuristic Information Update at each scheduling point in ES (RTMC)

```
1: if a task $\tau_i$ is arriving for the first time in the current hyper period then
2:     $pen(\tau_i) \leftarrow \lambda_1 \cdot |F(assign(\tau_i)) - F(SC^j(\tau_i))| + \lambda_3$ (cf. Equation (4.16))
3: end if
4: if there is a task $\tau_i$ causing an unbalanced execution in $[t_{last}, t_{current}]$ then
5:     $pen(\tau_i) \leftarrow pen(\tau_i) + \lambda_2 \cdot (t_{current} - t_{last})$
6: end if
7: $t_{last} \leftarrow t_{current}$
```

In Algorithm 14, the parameters $t_{last}$ and $t_{current}$ have the same meaning as in Algorithm 13.

Before the third activity DA is explained, there is a special treatment for the RTMC problem. The primary focus of the RTMC problem is on cluster-based multi-core platforms, where all the processor cores in the same cluster can only operate at a common frequency. The tasks running in parallel in a cluster must coordinate their frequencies. For this the ES-AS approach manages one process per cluster. The process decides the cluster-wide frequency at each scheduling point. Note that the scheduling points are differentiated on different processor cores in a cluster, even at the same time. For instance, if there are two tasks arriving or finishing simultaneously on two different cores, then they are treated as two different scheduling points. The cluster-wide frequency is selected as the highest one among all the required frequencies, because in this way it is able to guarantee that no task will finish later

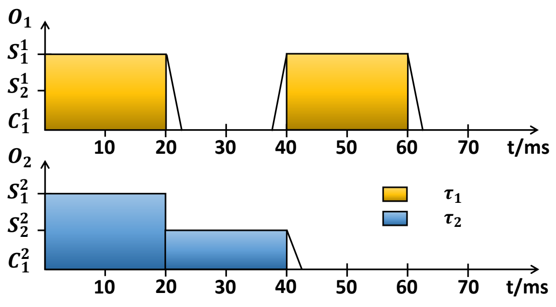


Figure 6.7: An example with two cores grouped in one cluster. $T(\tau_1) = 40$ ms, $T(\tau_2) = 80$ ms, $\tau_1$ is partitioned onto $O_1$ and $\tau_2$ onto $O_2$, $S_1^1$ is assigned to $\tau_1$ and $S_2^2$ to $\tau_2$

than its finishing time under worst case condition. On some platforms this strategy is even obliged due to hardware support [Intc]. Figure 6.7 shows one example, where $S_2^2$ is assigned to $\tau_2$. In the time interval $[0 \text{ ms}, 20 \text{ ms}]$ the task $\tau_2$ however is executed with $S_1^2$, because $S_1^1$ is required on the first core at the same time and is higher. Subsequently, the task $\tau_1$ completes its execution. Thus, in the time interval $[20 \text{ ms}, 40 \text{ ms}]$ the task $\tau_2$ is executed at its originally assigned frequency $S_2^2$. Clearly, $\tau_2$ completes earlier than the original completion time when it is constantly executed at $S_2^2$. In this manner, the system schedulability can be guaranteed.

If at a scheduling point one of the processor cores is going to be idle, it can be switched off to a low power state. This is the place where DA comes into play, because it takes care of the job of power state scheduling. The main challenge is to decide the arrival time $t$ of the next task. Since there is no I/O devices considered in the RTMC problem, the length of the upcoming idle time can be efficiently calculated. This is the main reason that the event recording is not required any more. In particular, $t$ is retrieved from a sorted queue $Q$. For each processor core $O_j$ there is a dedicated $Q^j$, which basically stores the arrival time of each task allocated on this core in sorted manner. Whenever a task is finished, it will update its next arrival time and reinsert it into $Q$. It is not hard to see that the time complexity of this process at each scheduling point is $O(\log n)$, where $n$ is the number of tasks in the system. More details about the queue management and its time complexity analysis can be found in Algorithm 15 and Lemma 6.4.1.

The variable $t_i$ in Algorithm 15 always stores the arrival time of the next job instance of the task $\tau_i$.

**Lemma 6.4.1.** *Algorithm 15 has a computation complexity of $O(\log n)$ at each scheduling point, where n is the total number of tasks in the system.*

*Proof.* It is obvious that $Q^j$ contains the arrival time of each task once at most, which means that the length of $Q^j$ can never be longer than $n$. By

**Algorithm 15** Manage task arrival time in a sorted queue and predict the length of idle time at each scheduling point in ES (RTMC)

```
1: if There exists a $\tau_i$ that is finishing on the processor core $O^j$ then
2:     $t_i \leftarrow t_i + T(\tau_i)$
3:     Reinsert $t_i$ into $Q^j$
4: end if
5: if A processor core $O_j$ becomes idle then
6:     Find the position of the current time $t_{curr}$ in $Q^j$
7:     Retrieve the next entry after that position. Let it be $t'$.
8:     Calculate the length of the next idle interval, which is $t' - t_{curr}$
9: end if
```

means of the binary search algorithm, an insertion operation or position finding takes $O(\log n)$ time. Hence, the computation complexity of Algorithm 15 is $O(\log n)$ at each scheduling point. □

As the length of idle time can be predicted, DA will shut down processor cores accordingly. Note that there is also a need to wake up the processor cores a little ahead of the actual arrival time of the next task, so that the task will not be delayed due to DPM state switching latency. The DPM state switching overhead is shown in Figure 6.7 by triangles. If the idle time of a processor core is shorter than the break-even time of all the low power states, the processor core needs to stay in its current P-state.

As a comparison, the application of ES-AS approach on the RTSC problem uses the event recording technique to compute the length of idle time, whereas hereby the idle time is directly predicted. Both methods have their advantages and disadvantages. The event recording yields a lower run-time overhead ($O(1)$), but needs to keep all the components staying active during the entire ES, which is clearly not energy efficient. The benefits and the drawbacks of direct idle time prediction is exactly the opposite. Furthermore, another reason why the event recording technique is chosen for the RTSC problem is that the device idle time prediction in a preemptive system at run-time is not computationally feasible [DA08a]. However, this is not the case for RTMC, because devices are not considered.

### 6.4.2 Application Stage

AS mainly applies the best solution found in ES. Hereby DA is taken as well to switch on and off processor cores accordingly.

In order to additionally exploit dynamic slack, a simple reclaiming mechanism is applied. More specifically, unused task execution time is utilized by

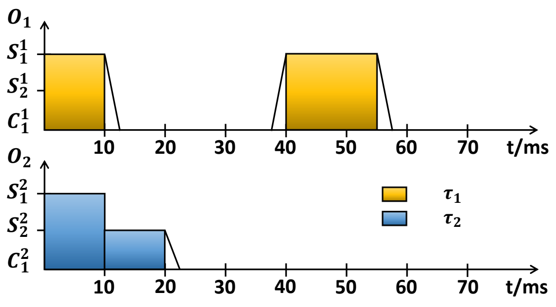


Figure 6.8: Dynamic slack reclaiming for DPM application

DPM to shut down processor cores earlier. Other sophisticated mechanisms, such as task procrastination, are not adopted here, because the online computation complexity is to be kept as small as possible. Figure 6.8 shows the similar scenario as in Figure 6.7, but the actual execution time of the first job instance of the task $\tau_1$ takes 10 ms. This leads to the situation that the task $\tau_2$ can run at $S_2^2$ already at 10 ms. By assuming that $\tau_2$ also completes earlier, then the cores can be switched off earlier as well.

The ES-AS approach is launched at system start and runs from ES to AS. As soon as there is a system change, such as a new task joining the system (only allowed at hyper period boundaries), the approach will start over with the calculation of a new initial solution. The new task is not accepted by the system, if no schedulable initial solution is possible. Clearly, the approach is extremely suitable for systems with long life time while system changes do not happen very often, because more hyper periods can be spent to obtain a better solution.

Figure 6.9 shows the activities taken by the ES-AS approach for the RTMC problem at a glance.

### 6.4.3 Correctness and Complexity

The proof of correctness of the ES-AS approach for the RTMC problem is similar to the case of the RTSC problem. At run-time, the system schedulability is always ensured due to three observations: i) only a schedulable solution is allowed to be applied in a hyper period, ii) no task will run below its assigned frequency, iii) and no task is delayed by DPM state switching overhead.

The total complexity of the ES-AS approach for the RTMC problem is shown in Theorem 6.4.1.

**Theorem 6.4.1.** *The proposed ES-AS approach has a computation complexity of $O(\log n + c)$ at each scheduling point and $O(n)$ at each hyper period boundary, where n is the number of tasks in the system and c is the number*

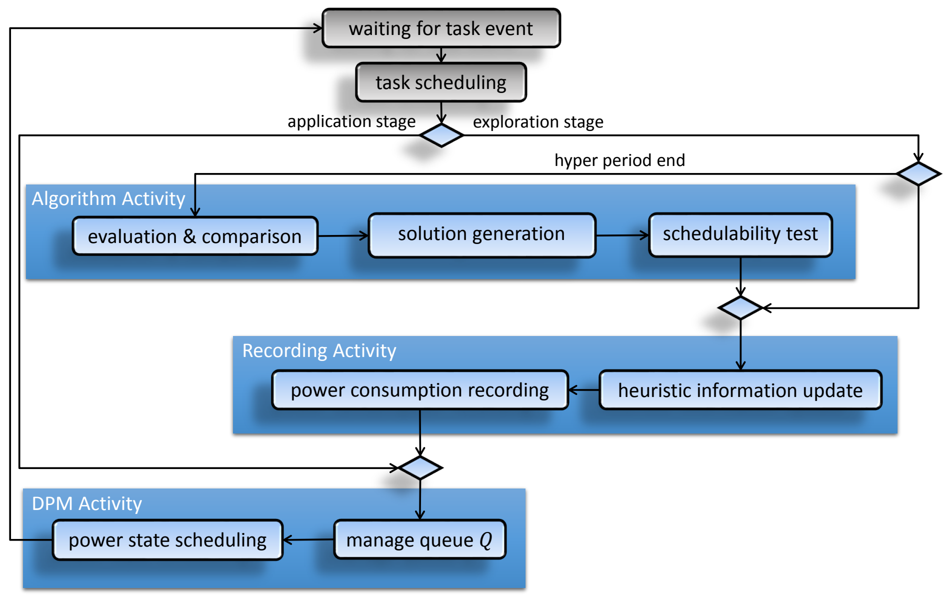


Figure 6.9: Activities performed in the ES-AS approach for the RTMC problem

*of supported processor low power states.*

*Proof.* The ES-AS approach contains two main parts. The first part happens at hyper period boundaries (AA), where the currently applied solution is evaluated and a new neighbor solution is generated. The evaluation of solution value is solved by run-time recording and the new solution generation needs to change the configuration of only one randomly selected task, which is depending on the heuristic information collected at run-time as well. Because the random number generation is based on non-uniform distribution, the neighbor generation takes $O(n)$. Afterward the schedulability test needs to only update the changed task utilization on the involved cores. Because the algorithm reallocates only one task from one core to another core at each hyper period boundary, there can be 2 processor cores getting involved at most. For these involved cores, the reward values need to be updated as well. Therefore this part takes $O(n)$ in total.

The second part occurs at each scheduling point and contains RA and DA as shown in Figure 6.9. The power consumption recording (Algorithm 13) and the heuristic information update take $O(1)$ for each processor core. Furthermore, according to Lemma 6.4.1 the effort to manage $Q$ and predict the length of idle time is $O(\log n)$ at each scheduling point. In case of a processor core being idle, DA needs to choose the most proper low power state. For this, it has to investigate the break-even time of all the low power states, which has a complexity of $O(c)$, provided the number of low power states is $c$. Thus the total complexity for RA and DA is $O(\log n + c)$ at each scheduling point. □

In practice, the number of low power states $c$ is usually relatively small and constant. Thus the complexity of the ES-AS approach at each scheduling point can be considered as $O(\log n)$.

## 6.5 Chapter Summary

This chapter introduced an online approach, the ES-AS approach, which is able to run the previously proposed HSASC and HSAMC algorithms in a fully adaptive fashion. The main concept is inspired by the general idea of Organic Computing, where a running system can adapt itself according to environment changes. The ES-AS approach is executed in two stages: Exploration Stage (ES) and Application Stage (AS). In brief, ES explores the solution space and tries to find the optimal solution. Hereby the iterations of the algorithms are mapped to the hyper periods of real-time systems. After ES is finished, AS simply applies the best solution found in ES. Base on this main concept, its application for the RTSC and RTMC problems is discussed in details. In addition, the correctness in terms of schedulability and the efficiency regarding run-time complexity of the proposed approach are analyzed.

In addition, the ES-AS approach provides several other advantages. First, this approach is independent on real-time scheduling algorithms. For instance, both EDF and RM can be applied here. Second, this approach is independent on optimization heuristics as well. Though this chapter shows the application for the HSASC and HSAMC algorithm, it, in general, could also be used with other meta-heuristics.

# Chapter 7

# Consideration of Non-Negligible DVS State Switching Overhead

Until now the proposed ES-AS approach has not explicitly taken DVS state switching overhead into account at run-time. Though its overhead is generally smaller than DPM state switching overhead [2], the reckless ignorance is not always safe in terms of timing constraints. The main contribution of this chapter is to improve the ES-AS approach by incorporating the handling of DVS overhead, especially for multi-core processor platforms. In particular, the traditional schedulability test is enhanced for the RTSC and RTMC problems concerning non-negligible DVS state switching overhead. The main part of this chapter was published in [HM12b] and [HM13b].

## 7.1 Introduction

The basic idea of DVS is to slow down active components by lowering their operating speed and voltage. Obviously, hereby a switching from one speed-voltage domain to another consumes time and energy. Voltage scaling is fairly expensive and is often in the same order of magnitude as a DPM state switching. For instance, a voltage switching on the Intel XScale [Inte] processor typically takes 0.45 ms and a shutdown to the deep-sleep mode takes 0.66 ms. If the additional management taken by an operating system is considered as well, DVS switching overhead becomes even higher. However, the state-of-the-art techniques, especially in terms of hard real-time systems, studied this problem rarely in depth. Most work either ignore this overhead or simply account it to the task WCET. However, as will be shown later, the latter is only applicable on single-core processor platforms.

[2]The main problem with DPM state switching overhead is to correctly predict the length of idle time. This is solved by the ES-AS approach through event recording activity (Algorithm 10) for the RTSC problem and the queue management process (Algorithm 15) for the RTMC problem.

The main difficulty on cluster-based multi-core processor platforms is accounted to the constraint that processor cores in the same cluster can only operate at the same speed. Some studies [AY03] and [CHK06] proposed to adapt single-core energy aware real-time scheduling algorithms to multi-core platforms by additionally performing a task partitioning in advance. The HSAMC algorithm proposed in this dissertation is belonging to this category as well. Hereby one of the major advantages is that well-established single-core real-time scheduling algorithms, such as EDF and RM, can be adopted for each core. In this way, the traditional schedulability test via utilization can be performed for each core independently. However, if cluster-based or full-chip multi-core processor platforms with non-negligible DVS state switching overhead are considered, this adaptation may not work appropriately any more, not only because of the reduced energy efficiency, but also due to the violation of timing constrains. As will be described in section 7.3, additional delays are introduced into the task execution, which makes traditional schedulability analysis insufficient. Section 7.3 proposes two solutions to enhance the schedulability analysis by taking DVS state switching overhead into consideration: the conservative protocol and the speed inheritance protocol.

In Chapter 3, the DVS state switching overhead is defined by using the terms $P(S_i,S_j)$ and $L(S_i,S_j)$ indicating the power consumption and the delay of switching between the states $S_i$ and $S_j$, respectively. As the main concerns are hard real-time systems and timing constraints, this dissertation concentrates on the switching latency. The power consumption of switching will be, however, briefly discussed in Chapter 9. Besides, in order to simplify the explanation, this work assumes that DVS switching overhead is uniformly equal, i.e., for any two P-states, the switching overhead between them is always the same, denoted by $L_p$. If a processor, nevertheless, prohibits a non-uniform switching overhead, the worst case (the longest latency) is used for $L_p$.

## 7.2 DVS Overhead Handling for RTSC

This section addresses the issue caused by DVS state switching overhead in the RTSC problem and proposes a solution by enhancing the traditional schedulability analysis.

### 7.2.1 Problem Description

On single-core processor platforms the problem caused by DVS state switching is quite straightforward. Since each task runs at its assigned speed, at each task switching point the core operating speed may have to be changed.

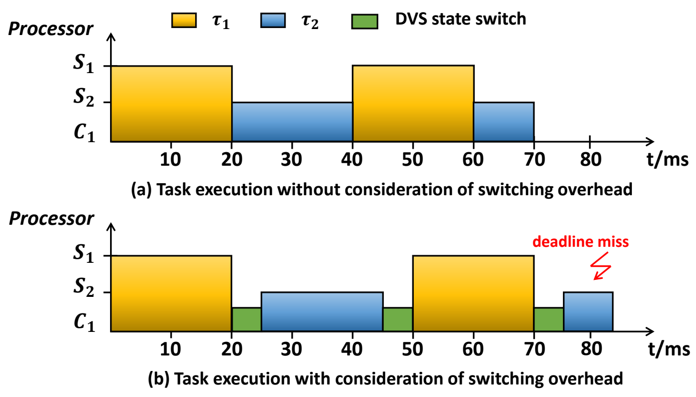


Figure 7.1: A comparison of task execution with and without DVS state switching overhead

Clearly, DVS switching latency introduces additional delays into the task execution. If this latency is not considered properly in schedulability analysis, some deadline misses may not be predicted. This problem is similar to the context switching problem. Figure 7.1 illustrates the comparison of task execution with and without consideration of DVS state switching overhead. This example assumes $W(\tau_1) = 20$ ms, $T(\tau_1) = 40$ ms, $W(\tau_2) = 15$ ms and $T(\tau_2) = 80$ ms. The tasks $\tau_1$ and $\tau_2$ are assigned with the P-states $S_1$ and $S_2$, respectively, where $F(S_1) = 1$ and $F(S_2) = 0.5$. The EDF algorithm is applied as real-time scheduler. Based on the traditional schedulability analysis, all the tasks can be scheduled without deadline miss.

$$U = \frac{W(\tau_1)\cdot F(S_1)}{T(\tau_1)\cdot F(S_1)} + \frac{W(\tau_2)\cdot F(S_1)}{T(\tau_2)\cdot F(S_2)} = \frac{1}{2} + \frac{3}{8} = \frac{7}{8} < 1 \tag{7.1}$$

If the DVS state switching overhead is ignored, as shown in Figure 7.1(a), all the tasks indeed complete before their deadlines. However, if the switching overhead is considered, which takes 5 ms in this example, the task $\tau_2$ misses its deadline at the time point 80 ms. The reason is rather obvious, because additional delays due to DVS state switches have been introduced into the task execution.

### 7.2.2 Enhanced Schedulability Analysis

In order to solve the above mentioned problem, this work provides Theorem 7.2.1 by taking DVS switching overhead into consideration in the schedulability analysis. Before the theorem is presented, an example is illustrated in Figure 7.2 to simplify the explanation.

This example involves 4 real-time tasks and assumes a priority based scheduling algorithm. In terms of task priority, $\tau_1 < \tau_2 < \tau_3 < \tau_4$ holds. During

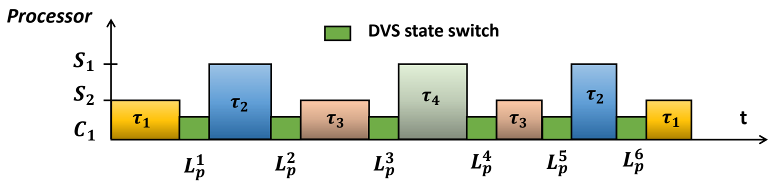


Figure 7.2: A single-core processor example with 4 real-time tasks

execution, one further assumes that $\tau_1$ is preempted by $\tau_2$. $\tau_2$ is in turn preempted by $\tau_3$, which is again preempted by $\tau_4$. As a result, there are in total 6 DVS state switches ($L_p^1$, $L_p^2$, etc.) shown in the figure. Note that a DVS switch is always associated with a context switch. In other words, if there is a DVS state switch then there must be a context switch. However, a context switch may not be necessarily accompanied with a DVS state switch.

Therefore, in order to take DVS state switching overhead into account, it is sufficient to count the number of context switches [Dev03]. In a preemptive real-time system, context switches can be classified into two categories according to their causes:

1. Switches due to preemption: These context switches are cased by preemption when a task with higher priority arrives while the currently running task has a lower priority. In Figure 7.2, the switches $L_p^1$, $L_p^2$ and $L_p^3$ are clearly belonging to this category.
2. Switches due to task completion: These context switches are caused when a task completes and there is at least one ready task is waiting for execution. Obviously, the switches $L_p^4$, $L_p^5$ and $L_p^6$ in Figure 7.2 fall in this category.

Based on this classification, context switches can be charged to tasks. In particular, a switch due to preemption is charged to the task, which has caused the preemption, i.e., the task which just arrives and has a higher priority than the current running task. A switch due to task completion is charged to the task, which just completes. Following this rule, it is not hard to see that a task may be responsible for at most one switch due to preemption and one switch due to task completion inside its period, because the task arrives and completes only once. Consequently, a task may cause two context switches at most, once at the arriving time and once at the completion time. In Figure 7.2, clearly $L_p^1$ and $L_p^6$ are accounted to $\tau_2$, $L_p^2$ and $L_p^5$ are accounted to $\tau_3$ and finally $L_p^3$ and $L_p^4$ are accounted to $\tau_4$. Except the task $\tau_1$, all the tasks are causing exactly two switches. If the example is extended to running $n$ tasks, the required switches become $2n-2$.

Now it is ready to introduce Theorem 7.2.1.

**Theorem 7.2.1.** *A hard real-time system on a DVS enabled single-core processor platform is schedulable, if the following condition is satisfied:*

$$\sum_{i=1}^{n} \frac{\frac{W(\tau_i) \cdot F(S_1)}{F(assign(\tau_i))} + 2 \cdot L_p}{T_i} \leq U_B \tag{7.2}$$

*Proof.* The delays are always caused by P-state switching, which may only happen at context switch. Since each task may cause two context switches at most inside its period, the maximal number of delays caused by a job is limited by two. If these two delays are accounted into task worst case execution time, clearly the system feasibility can be guaranteed. □

If the processor utilization in the previous example (shown in (7.1)) is recomputed using (7.2), the deadline miss can be predicted.

$$U = \frac{20\text{ ms} + 2 \cdot 5\text{ ms}}{40\text{ ms}} + \frac{30\text{ ms} + 2 \cdot 5\text{ ms}}{80\text{ ms}} = \frac{5}{4} > 1 \tag{7.3}$$

Clearly, Theorem 7.2.1 gives a conservative analysis of processor utilization. In general, the test (7.2) overestimates the real utilization of processor. This is mainly due to two reasons: i) not every task causes exact two context switches and ii) not every context switch is accompanied with a DVS state switch and thus a delay. However, the system schedulability is of utmost importance in a hard real-time system. The analysis must work on all the scenarios, even in the worst case.

## 7.3 DVS Overhead Handling for RTMC

As shown in Chapter 4, the HSAMC algorithm is belonging to the category of partitioned scheduling. The schedulability analysis can be performed on each processor core independently. Thus, the problem described in the previous section applies here as well. However, due to the special property of cluster-based multi-core processor platforms, the previously proposed enhanced schedulability analysis is not sufficient enough. This section addresses these issues and proposes two solutions: the conservative protocol and the speed inheritance protocol.

### 7.3.1 Problem Description

In order to better explain, the problem is illustrated by a simple example shown in Figure 7.3. The processor is equipped with two processor cores grouped in one cluster. Both processor cores share the same power model and support two P-states with $F(S_1^1) = F(S_1^2) = 1$ and $F(S_2^1) = F(S_2^2) = 0.5$. The switching latency between the two states is 5 ms. Moreover, there are two real-time tasks, which are partitioned to the first core and the second core, respectively. The task $\tau_1$ is assigned with $S_2^1$ and the task $\tau_2$ is assigned with $S_1^1$. The tasks $\tau_1$ and $\tau_2$ have a speed conflict during the time interval between 0 ms and 5 ms, because they run in parallel but require different operating speeds. The strategy is to select the highest speed as the cluster-wide operating speed. Taking the schedulability test from Theorem 7.2.1, the utilization of the processor cores can be computed.

$$U(O_1) = \frac{80\text{ ms} + 2 \cdot 5\text{ ms}}{100\text{ ms}} = \frac{9}{10} < 1 \tag{7.4}$$

$$U(O_2) = \frac{5\text{ ms} + 2 \cdot 5\text{ ms}}{20\text{ ms}} = \frac{3}{4} < 1 \tag{7.5}$$

According to this result, the tasks are clearly schedulable on both processor cores, provided that the EDF scheduling algorithm is applied. However, in the actual task execution (cf. Figure 7.3) the task $\tau_1$ will miss its deadline at 100 ms. The previously proposed schedulability test is obviously not sufficient any more for the RTMC problem. This situation is quite plausible, because additional delays are introduced into the execution of the task $\tau_1$. For instance, at 5 ms the task $\tau_2$ is finished and the task $\tau_1$ can run at its originally assigned speed, however, it will first take 5 ms to switch the speed from $S_1^1$ to $S_2^1$. At 20 ms, the task $\tau_2$ becomes active again and will require $S_1^2$, it will also need 5 ms to switch the speed from $S_2^2$ back to $S_1^2$. Note that $\tau_1$ sometimes runs at a higher than its assigned speed, one may expect that $\tau_1$ will finish earlier than its WCET. However, this will only shorten the task execution time but not the completion time. In this example, there are too many delays caused by P-state switching that can not be compensated by the shortening of execution time. One important observation here is that these delays are caused by the so-called Inter-Core Preemption (ICP), which is generated by tasks from other cores in the same cluster. Formally, ICP is defined as follows:

**Definition 7.3.1 (ICP).** *An Inter-Core Preemption of a task $\tau_i$ is a preemption of its execution due to core speed change, which is caused by the arrival or the completion of another task $\tau_j$ with the conditions:*

$$group(alloc(\tau_i)) = group(alloc(\tau_j)) \tag{7.6}$$

$$alloc(\tau_j) \neq alloc(\tau_i) \tag{7.7}$$

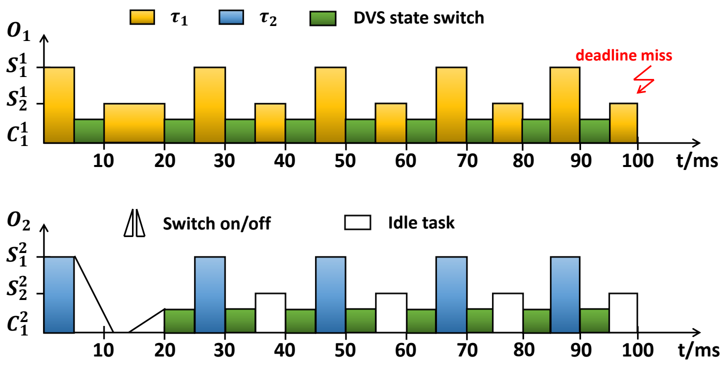


Figure 7.3: An example with two tasks running on a platform with two cores grouped into one cluster. Both processor cores support two operating speeds with the switching latency $L_p = 5$ ms, $W(\tau_1) = 40$ ms, $W(\tau_2) = 5$ ms, $T(\tau_1) = 100$ ms, $T(\tau_2) = 20$ ms, $alloc(\tau_1) = O_1$, $alloc(\tau_2) = O_2$, $assign(\tau_1) = S_2^1$ and $assign(\tau_2) = S_1^2$.

### 7.3.2 Enhanced Schedulability Analysis

The following sections present solutions to enhance the schedulability analysis by taking non-negligible DVS state switching overhead into consideration.

#### Conservative Protocol (CP)

As mentioned earlier in the problem description, the main cause of deadline misses is due to additional delays introduced by ICPs. Therefore, the most straightforward idea is to estimate the maximal number of such delays for each task and account them into their task WCETs. Formally, Theorem 7.3.1 defines the schedulability test of the conservative protocol.

**Theorem 7.3.1.** *Given a cluster-based multi-core processor platform and a set of independent periodic real-time tasks, a solution of the RTMC problem containing a task partition and a speed assignment can guarantee the system feasibility, if the following condition for each core $O_j$ holds:*

$$\sum_{alloc(\tau_i)=O_j} \frac{\frac{W(\tau_i)\cdot F(S_1^j)}{F(assign(\tau_i))} + 2\cdot L_p + num_i \cdot L_p}{T(\tau_i)} \leq U_B \tag{7.8}$$

*where $num_i$ is the maximal number of delays that can be introduced into the execution of the task $\tau_i$ due to ICPs. Formally it is computed by (7.9).*

$$num_i = \sum_{\tau_j} \lceil \frac{T(\tau_i)}{T(\tau_j)} \rceil \cdot 2,\ \tau_j\ satisfies\ cond.\ in\ Definition\ 7.3.1 \tag{7.9}$$

*Proof.* The proof starts with the correctness of (7.9), which estimates the maximal number of delays for each task $\tau_i$. Clearly, only the tasks satisfying the conditions in Definition 7.3.1 may generate ICPs for $\tau_i$. Additionally, it is obvious that the number of ICPs generated by a task $\tau_j$ is limited by the number of its occurrence during the period of the task $\tau_i$. Thus, for each task the maximal number of its occurrence can be computed by the term $\lceil \frac{T(\tau_i)}{T(\tau_j)} \rceil$. Since each ICP may cause two delays at most (as shown in Figure 7.3), once at the arriving and once at the completion, the maximal number of delays is equal to the number of occurrence multiplied with 2. Furthermore, in order to cover the problem that already exists on single-core processor platforms, Theorem 7.2.1 is applied. As a result, a sufficient condition for the schedulability analysis is obtained in (7.8). □

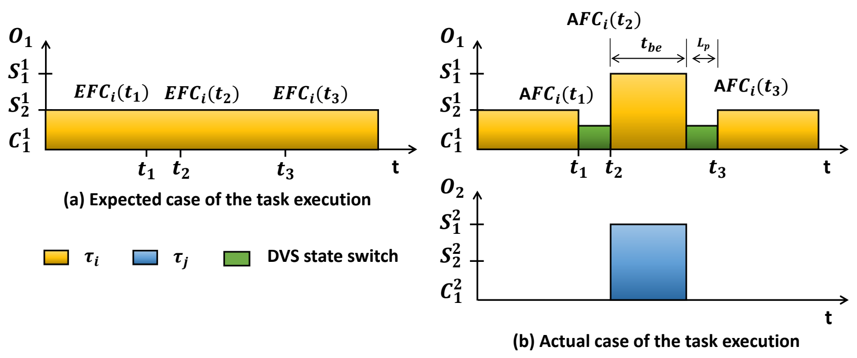
If the schedulability analysis of the conservative protocol is applied to the example shown in Figure 7.3. The utilization on the first processor core $O_1$ is computed as follows.

$$U(O_1) = \frac{80\text{ ms} + 2 \cdot 5\text{ ms} + 10 \cdot 5\text{ ms}}{100\text{ ms}} = \frac{7}{5} > 1 \tag{7.10}$$

This clearly indicates that the system is not feasible. Theorem 7.3.1 gives a sufficient condition for the schedulability test. It is not hard to see that this test is rather pessimistic. It is not necessary that each occurrence will generate an ICP. Many solutions may fail the test, even though they are indeed schedulable. This is also the reason why this approach is called conservative protocol. However, one of the advantages by this protocol is that there is no need to modify task execution. In the next step a more sophisticated approach is presented, where the test is much tighter.

#### Speed Inheritance Protocol (SIP)

The speed inheritance protocol is based on an above mentioned observation. In Figure 7.3, due to ICPs the task $\tau_1$ runs occasionally at a higher than its assigned speed. Thus, the execution time of $\tau_1$ is shorter than its original WCET. However, $\tau_1$ still missed its deadline, because too many delays are introduced and the amount of saved execution time is not enough to break even the delays. The basic idea of SIP is then to further shorten the execution time of such tasks, so that the delays can be compensated. In sum, each time a task is interrupted by an ICP, its operating speed is usually increased. Therefore, the task will also run faster. The speed inheritance protocol utilizes this property to compensate the delays caused by ICPs by enforcing the



$O_1$
$S_1^1$
$S_2^1$
$C_1^1$
$EFC_i(t_1)$ $EFC_i(t_2)$ $EFC_i(t_3)$
$t_1$ $t_2$ $t_3$ t
(a) Expected case of the task execution
$\tau_i$ $\tau_j$ DVS state switch
$AFC_i(t_2)$
$t_{be}$
$L_p$
$AFC_i(t_1)$
$AFC_i(t_3)$
$O_2$
$S_1^2$
$S_2^2$
$C_1^2$
(b) Actual case of the task execution

Figure 7.4: Task execution in the expected and actual case. $assign(\tau_i) = S_2^1$, $assign(\tau_j) = S_1^2$ and $\tau_j$ arrives at $t_2$

task running at that higher speed for a certain amount of time. This idea is similar to the DPM break-even time. In this way the delays do not need to be accounted into task WCETs any more. The name, speed inheritance, comes from the fact that this protocol tries to run tasks at a higher speed inherited from other tasks.

In order to calculate the break-even time, there is a need to analyze how much the execution of a task is delayed. For this, the definition of finished cycles of a task is introduced. Hereby one distinguishes the expected case and the actual case.

**Definition 7.3.2.** *$EFC_i(t)$ is defined as the Expected Finished Cycles of task $\tau_i$ until a time point $t$. This describes the finished cycles of a task until $t$ in the expected case where no ICP happens.*

**Definition 7.3.3.** *$AFC_i(t)$ is defined as the Actually Finished Cycles of task $\tau_i$ until a time point $t$. This describes the finished cycles of a task until $t$ in the actual case of task execution.*

Figure 7.4 shows the execution of a task $\tau_i$ in the expected and actual case, respectively. The expected case reflects the task execution as planned. If all the tasks are executed as expected, then clearly no task will miss its deadline. However, due to ICPs a task execution may vary in the actual case. For instance, at the time point $t_1$ in Figure 7.4 $EFC_i(t_1) = AFC_i(t_1)$ clearly holds, because no delay has been introduced before $t_1$. However, at $t_2$, $AFC_i(t_2)$ is already behind $EFC_i(t_2)$, as no work has been done during the interval between $t_1$ and $t_2$ in the actual case. The main idea here is to compute the time $t_{be}$ needed at least to compensate the delays. In other words, the task $\tau_i$ needs to run at the higher speed $S_1^1$ for at least $t_{be}$, so that the actual case execution can catch up the progress in the expected case. Formally, the equation (7.11) is to be ensured.

$$AFC_i(t_3) = EFC_i(t_3) \tag{7.11}$$

Moreover, the equations (7.12) and (7.13) compute the actually and expected finished cycles of $\tau_i$ until $t_3$, respectively. According to their definitions, they are equal to the already finished cycles until $t_2$ plus the cycles executed in $[t_2, t_3]$. Note that no cycles are executed during DVS state switching in the actual case and $t_3 - t_2 = t_{be} + L_p$ holds.

$$AFC_i(t_3) = AFC_i(t_2) + t_{be} \cdot F(S_1^1) \tag{7.12}$$

$$EFC_i(t_3) = EFC_i(t_2) + (t_{be} + L_p) \cdot F(S_2^1) \tag{7.13}$$

By applying (7.12) and (7.13) in (7.11), the formula to compute the break-even time $t_{be}$ is obtained.

$$t_{be} = \frac{EFC_i(t_2) - AFC_i(t_2) + L_p \cdot F(S_2^1)}{F(S_1^1) - F(S_2^1)} \tag{7.14}$$

The speed inheritance protocol does not modify the actual scheduling algorithm, but rather has the impact on the selection of task execution speed. Now the additional effort is described that has to be taken at run-time to enable the speed inheritance protocol. The $AFC$ and $EFC$ values for each task are initialized with 0 at task arrival and updated at each scheduling point. Once a task is interrupted by an ICP, $t_{be}$ needs to be computed by means of (7.14). The system ensures that this task runs at the higher speed at least for $t_{be}$. Afterward the task can run at its originally assigned speed again. If an intra-core preemption occurs during this time interval, which is caused by a task with higher priority in the same processor core, then the system remembers the time that has been consumed and the remaining time will be consumed after the task is resumed. However, if another ICP happens and the task operating speed is increased again, then there is a need to recalculate $t_{be}$ and ensure the task runs at the new speed for $t_{be}$. Before deriving the schedulability test, Lemma 7.3.1 shows one important feature of the $EFC_i(t)$ and $AFC_i(t)$ values for each task $\tau_i$, i.e., the maximal cycles that the $AFC_i$ value may lag behind the $EFC_i$ value is limited.

**Lemma 7.3.1.** *If a task $\tau_i$ is running on $O_k$, then the following condition holds at any time point $t$* [3]*:*

$$EFC_i(t) - AFC_i(t) \leq num_i^{'} \cdot L_p \cdot F'(assign(\tau_i)) \tag{7.15}$$

*where*

$$num_i^{'} = |\{S_j^k | F(S_j^k) > F(assign(\tau_i)) \ and \ F(S_j^k) \leq F(S_1^k)\}| \tag{7.16}$$

[3]Hereby $F'(S)$ denotes the real frequency in MHz of the P-state $S$. This is unlike $F(S)$, which gives the normalized frequency with regard to the full performance state $S_1$. More explanation is presented in the definition of single-core processor power model in Chapter 3.

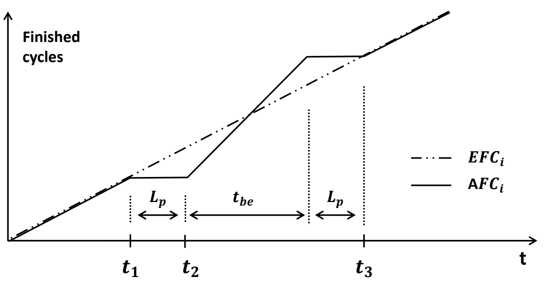


Figure 7.5: The relationship between $EFC$ and $AFC$ based on the example in Figure 7.4.

*Proof.* At the task arrival, the $EFC_i$ and $AFC_i$ values are clearly equal, since they are initialized with 0. This equality will not be broken, unless an ICP occurs. The curves in Figure 7.5 illustrate the value trend of $EFC_i$ and $AFC_i$ based on the example shown in Figure 7.4. Obviously, at the time point $t_2$, $AFC_i$ lags behind $EFC_i$ the most. Afterward $AFC_i$ starts to catch up $EFC_i$, because the task $\tau_i$ runs now at a higher speed. In order to find the maximal value of $EFC_i - AFC_i$, it requires to consider the worst case, i.e., several ICPs happen consecutively. For instance in Figure 7.4, if at $t_2 + \varepsilon$ ($\varepsilon$ is an indefinitely small positive value) a third task interrupts the task $\tau_i$ due to ICP and requires an even higher speed, then the distance between the $EFC_i$ curve and the $AFC_i$ curve will become larger. However, since each of these ICPs for $\tau_i$ is always accompanied with the speed increase of the task $\tau_i$, the number of the consecutive ICPs is limited by the number of available speeds, which is expressed in (7.16). Finally, if the time delays are transformed into cycles, the condition (7.15) is obtained. □

In Figure 7.5, one assumes that there are enough cycles to be executed at higher speed and therefore delays can be compensated. However, it may happen that an ICP occurs at the near end of a task execution and the remaining cycles are not sufficient. To overcome this problem, Theorem 7.3.2 provides a solution which as well serves as a schedulability test for the speed inheritance protocol.

**Theorem 7.3.2.** *Given a cluster-based multi-core processor platform and a set of independent periodic real-time tasks, a solution of the RTMC problem containing a task partition and a speed assignment can guarantee the system feasibility under the speed inheritance protocol, if the following condition for each core $O_j$ holds*

$$\sum_{alloc(\tau_i)=O_j} \frac{\frac{W(\tau_i)\cdot F(S_1^j)}{F(assign(\tau_i))} + 2\cdot L_p + num_i' \cdot L_p}{T(\tau_i)} \leq U_B \qquad (7.17)$$

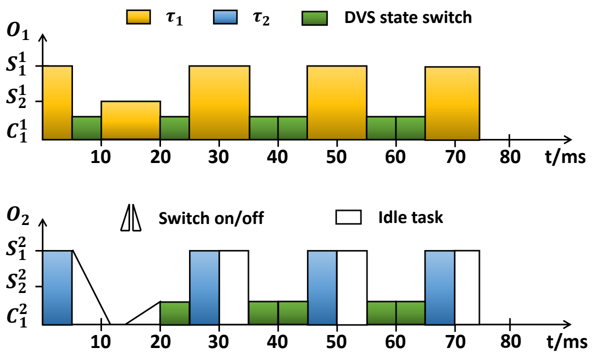


Figure 7.6: The example running under SIP

*Proof.* By applying Lemma 7.3.1 the maximal cycles that $AFC_i$ may lag behind $EFC_i$ can be obtained. In most cases this lag will be compensated by using the speed inheritance technique. However, it is under the assumption that a task has enough time to do it, i.e., the remaining cycles of the task are enough for $t_{be}$. In other words, if there is not enough time, there is no chance to compensate these delays. In this case, SIP uses the old fashion where these non-compensable delays are accounted to task WCETs. Therefore, a sufficient condition (7.17) for the schedulability test by using the speed inheritance protocol is obtained. □

If the example shown in Figure 7.3 is re-considered with the application of SIP, the task execution looks like as in Figure 7.6, where all the tasks can be scheduled without any deadline miss. Hereby the time that $\tau_1$ runs at the higher than its assigned speed is longer than in Figure 7.3. That is, the saved execution time is now enough to compensate the delays due to ICPs.

Clearly, the speed inheritance protocol makes a tighter bound than the conservative protocol, because the number of delays added into task WCETs is now limited by the number of available speeds, which is usually relatively small. However, the speed inheritance protocol requires run-time support by maintaining the $EFC$ and $AFC$ values as mentioned above.

## 7.4 Chapter Summary

This chapter considered explicitly the non-negligible overhead of DVS state switching. The problem is first investigated for single-core processor platforms and subsequently extended for cluster-based multi-core processor platforms. In particular, two protocols with their enhanced schedulability analysis are provided. The conservative protocol requires no run-time modification of task execution, however, its schedulability test is rather pessimistic. In contrast, the speed inheritance protocol bounds the test much tighter but needs additional run-time management effort. In total, both protocols provide sufficient condition test for schedulability analysis.

# Chapter 8

# Evaluation

In this chapter, the previously proposed approaches are thoroughly evaluated. The first section introduces an overview of evaluation objectives. Afterward, a more detailed description of evaluation results is presented. In general, the evaluation work is divided into two parts. The first part concentrates on synthetic test scenarios, where a large number of randomly generated test inputs are studied through simulation in order to obtain statistically significant results. In the second part, real-life case studies are investigated using real target platforms to prove the practical applicability. Finally, the last section concludes this chapter.

## 8.1 Objectives

As a remainder, the main contributions of the dissertation are the HSASC and HSAMC algorithms as well the ES-AS approach allowing them to be executed in an online fashion. Consequently, the first objective is to evaluate energy reduction efficiency of the algorithms, i.e., how much energy the proposed algorithms can save, especially in comparison with existing techniques. Since the proposed algorithms are running online by means of the ES-AS approach, the run-time plays an important role. Therefore, the second objective is to evaluate the duration of the algorithms based on the proposed termination criterion, i.e., how long the exploration stage takes. The termination criterion is derived using regression technique, which provides an estimated measurement of algorithm performance. In order to justify the correctness and efficiency of the termination criterion, the third objective of the evaluation is to investigate estimation accuracy. Moreover, though an asymptotic behavior of run-time overhead of the ES-AS approach is given in Chapter 6, the absolute overhead is quite interesting as well, because the theoretical results do not care about the constants. The forth objective is to evaluate the real run-time overhead introduced by the ES-AS approach. Furthermore, two extensions of the real-time scheduling analysis are proposed to deal with non-negligible DVS state switching overhead. The fifth objective is evaluation of the impact of the conservative protocol and speed inheritance protocol. Finally, in order to prove the practical applicability of the proposed approaches, evaluations are conducted on real target platforms using real-life case studies. The main objectives are summarized as follows.

- O1: Energy reduction efficiency of the HSASC and HSAMC algorithms
- O2: Run-time of the HSASC and HSAMC algorithms
- O3: Accuracy of the performance estimation using regression technique
- O4: Run-time overhead of the ES-AS approach
- O5: Impact of the conservative protocol and the speed inheritance protocol
- O6: Practical applicability

For the objectives O1, O2, O3, O4 and O5, the evaluation is performed in a simulation framework by running synthetic task sets, which are randomly generated. The objective O6 is achieved by implementing the approaches in real target platforms. In what follows, they are described in detail.

## 8.2 Synthetic Test Scenarios

Before evaluation results are presented, the experiment setup will be first explained. As this evaluation process involves simulation and synthetic task sets, the following two subsections introduce some general information about the simulation framework and the random task set generator.

### 8.2.1 Abstract RTOS Simulation Framework

As an execution environment, this work adopted an abstract Real-Time Operating System (RTOS) simulation framework described in [ZMG09]. The simulator is implemented using SystemC, which is a C++ based system level design language. SystemC [Ass] was initiated by the Open SystemC Initiative (OSCI) consortium (now merged with Accellera [Ini]) and later approved as an IEEE standard. Technically, SystemC is implemented as a C++ library providing an event-driven simulation semantic. The primary usage of the SystemC language is for system-level modeling and simulation.

The basic idea of abstract RTOS simulation is to utilize the advantage of SystemC to omit unnecessary information and only concentrates on real-time scheduling aspects. More specifically, the abstraction is mainly made in two aspects:

- The actual task execution on a target processor is abstracted to the simulation of required execution time.
- The details of RTOS behavior, such as task management, memory management and resource management, are abstracted to a real-time scheduler, which dispatches real-time tasks at the proper time.

The key benefit of this abstraction enables early evaluation of scheduling behavior, i.e., the impacts and effects of different real-time scheduling algorithms can be investigated without knowing a concrete target processor architecture and RTOS in advance. This feature perfectly matches the needs of the evaluation process in the context of this dissertation, because the proposed approaches are independent of concrete target platforms. As inputs, the abstract RTOS simulator accepts a set of software tasks with specified real-time properties including deadlines, periods and task WCETs. The well-established real-time scheduling strategies like EDF and RM are already implemented in the simulation framework and can be simply selected through a configuration file. As outputs, the simulator produces, on one side, a human readable log file and on the other side a Value Change Dump (VCD) [Bey] file of signal traces. The VCD file can be shown graphically using popular tools like GTKWave [GTK].

Unfortunately, the original abstract RTOS simulator lacks the ability of power aware simulation, i.e., DPM and DVS are not available. Therefore, some extensions and modifications are made in the implementation. In short, two C++ classes are essentially declared for processors and devices, respectively. For each concrete processor or device, an object of the corresponding class with proper power characteristics is created at run-time. In fact, the objects represent the power model of processors and devices. As shown in Chapter 3, the power model is defined in form of a finite state machine, based on which the behavior part of the objects is implemented. For instance, there is a state variable inside the processor class always indicating the current running frequency. This variable can be changed by a power aware real-time scheduler and the respective task WCETs need to be scaled up or down accordingly. In case of the device class, there is a variable representing the current DPM state. Moreover, as this dissertation deals with non-negligible state switching overhead, they are implemented as a delay of simulation time.

In order to make the evaluation more realistic, the power model of processors and devices are not arbitrarily generated, but rather taken from either data

| State | $F$ | $P$ (mW) | $T_{on \to off}$ | $T_{off \to on}$ | $P_{on \to off}$[4] | $P_{off \to on}$[4] |
|---|---|---|---|---|---|---|
| $S_1$ | 1[5] | 925 | n.a.[6] | n.a. | n.a. | n.a. |
| $S_2$ | 0.83 | 747 | n.a. | n.a. | n.a. | n.a. |
| $S_3$ | 0.67 | 570 | n.a. | n.a. | n.a. | n.a. |
| $S_4$ | 0.5 | 390 | n.a. | n.a. | n.a. | n.a. |
| $S_5$ | 0.33 | 279 | n.a. | n.a. | n.a. | n.a. |
| $C_1$ | n.a. | 15.4 | 1 µs | 1 µs | 110 mW | 110 mW |
| $C_2$ | n.a. | 1.72 | 340 µs | 430 µs | 100 mW | 100 mW |
| $C_3$ | n.a. | 0.16 | 560 µs | 500 µs | 90 mW | 90 mW |
| $C_4$ | n.a. | 0.1 | 660 µs | 600 µs | 90 mW | 90 mW |

Table 8.1: The power model of the Intel XScale® processor [Inte]

| State | $F$ | $P$ (mW) | $T_{on \to off}$ | $T_{off \to on}$ | $P_{on \to off}$[4] | $P_{off \to on}$[4] |
|---|---|---|---|---|---|---|
| $S_1$ | 1[7] | 999.9 | n.a. | n.a. | n.a. | n.a. |
| $S_2$ | 0.70 | 548.64 | n.a. | n.a. | n.a. | n.a. |
| $S_3$ | 0.17 | 90.55 | n.a. | n.a. | n.a. | n.a. |
| $C_1$ | n.a. | 1.94 | 163 µs | 163 µs | 60 mW | 60 mW |
| $C_2$ | n.a. | 0.095 | 2852 µs | 2852 µs | 60 mW | 60 mW |

Table 8.2: The power model of ARM Cortex™-A8 [OMA][Insa]

sheets or other related studies. The processors used in the test are the Intel XScale® processor and the ARM Cortex™-A8 processor. The power model of the former, which is shown in Table 8.1, is taken from [Inte]. The power model of the latter, which is shown in Table 8.2, is taken from [OMA] and [Insa]. In fact, this model is the one used in OMAP3530 [Insb] from Texas Instruments. More concretely, [Insa] provides a spreadsheet, which is able to estimate chip power consumption in different DVS and DPM states. Note that the OMAP3530 chip contains more than one power domain and the most interesting part is the MPU power domain, which mainly contains the ARM core. Furthermore, the state switching latency is obtained from [OMA].

In order to build a cluster-based multi-core processor, the power model of the two aforementioned processors are adopted and simply multiplied. In other words, each processor core either has the power model of the Intel XScale® processor or the power model of the ARM Cortex™-A8 processor. Formally, they are referred to as I-core and A-core, respectively. Note that the processor cores inside a cluster must share the same power model. To simplify the evaluation, this work concentrates on the symmetry architecture,

[4]This value is not given in the data sheet, therefore it is arbitrarily chosen between the power consumption of the active state and the low power state.

[5]The operating frequency without normalization is 624 MHz.

[6]not applicable.

[7]The operating frequency without normalization is 600 MHz.

| Device | $P(D_0)$ | $P(D_1)$ | $T_{on\to off}/T_{off\to on}$ | $P_{on\to off}/P_{off\to on}$ |
|---|---|---|---|---|
| MSWM | 750 mW | 5 mW | 40 ms | 100 mW |
| IBMM | 1300 mW | 100 mW | 12 ms | 500 mW |
| SSTF | 125 mW | 1 mW | 1 ms | 50 mW |
| STFC | 225 mW | 20 mW | 2 ms | 100 mW |
| REC | 190 mW | 85 mW | 10 ms | 125 mW |
| FHD | 2300 mW | 1000 mW | 20 ms | 1500 mW |

Table 8.3: The power model of applied I/O devices [CG06]

i.e., each cluster contains the same number of processor cores and the number of clusters is power of two. However, it is worth mentioning that the proposed approaches are not only constraint to the symmetry architecture.

A quad-core processor with two clusters may be built by using two I-cores in the first cluster and two A-cores in the second cluster. This multi-core processor is denoted by `I2A2`. In general, the letters indicate the type of processor cores in the cluster and the digits give the size of the cluster. To further simplify the notation, same clusters are merged and its number is denoted by index. For instance, $(\texttt{I1})_4$ denotes a platform with 4 cores in total that are grouped into 4 clusters, i.e., a per-core platform and all the cores are I-cores. In this evaluation work, the investigated variants are shown below.

- Processor with 2 cores: $(\texttt{I1})_2$ and `I2`
- Processor with 4 cores: $(\texttt{I1})_4$, $(\texttt{I2})_2$, `I2A2` and `I4`

Since this dissertation addresses system-wide power consumption, where I/O peripherals are taken into consideration, the following devices are applied in the evaluation scenario.

- MaxStream Wireless Module (MSWM)
- Realtek Ethernet Chip (REC)
- IBM Microdrive (IBMM)
- SST Flash SST39LF020 (SSTF)
- SimpleTech Flash Card (STFC)
- Fujitsu 2300AT Hard Disk (FHD)

Their power models are taken from [CG06] and shown in Table 8.3.

| Platforms | Number of tests | Size | Utilization |
|---|---|---|---|
| With 2 cores | 1600 | $[5,12]$ | $[0.4,1.6]$ |
| With 4 cores | 1600 | $[7,14]$ | $[0.8,3.2]$ |

Table 8.4: Properties of generated task sets for multi-core processor platforms

### 8.2.2 Generation of Synthetic Task Sets

Another important element in the synthetic test scenario is task set, which serves as inputs of the abstract RTOS simulation framework. In order to obtain statistically significant results, a large number of task sets need to be generated. Hereby an open source tool Task Graph For Free (TGFF) [RDV] is applied to accomplish this job. The main intend of TGFF is to facilitate users to generate pseudo-random test benches for task scheduling and resource allocation related research. Its usage can be found in a large number of studies. For instance, some of them are [FN08], [CYW10], [Cha+12], [Gan+12] and [Wan+12]. In general, TGFF is able to produce a set of tasks as a collection of task graphs, each of which is expressed by a Directed Acyclic Graph (DAG). In other words, each task graph represents a set of dependent tasks and a complete task set is composed of several task graphs. In addition, TGFF generates WCET for each task and period/deadline for each task graph. Note that all the tasks inside a DAG share the same deadline and period. Since this work concentrates on independent tasks, all the tasks in a task graph (DAG) are merged into one real-time task. A DAG usually is in form of a tree-like structure. TGFF provides a configuration possibility that the maximum in-degree and out-degree of task nodes in a DAG can be defined. Thus, setting them to 1 results in a directed linear task graph, in which all the tasks are then combined into a single real-time task. The WCET of the new task is the sum of generated WCETs of all the tasks in the DAG and the period remains as the same. Furthermore, the deadline of each new task is assumed to be equal to the period.

In case of the RTSC problem, 1600 task sets are randomly generated. The size of each produced task set is within $[5,12]$. More specifically, there are 200 task sets generated in average for each size. The period of each task is randomly chosen in the range of $[5\text{ ms}, 150\text{ ms}]$. The processor utilization of each task set is uniformly distributed in $(0,1)$. Furthermore, for each task $0 \sim 2$ devices are randomly selected from Table 8.3 as the required devices.

The case of RTMC is slightly more complicated and more task sets need to be generated. Since the investigated multi-core processor platforms can be classified into two categories according to the number of cores, i.e., processor with 2 cores or with 4 cores, 1600 task sets are randomly generated for each type of them. In general, a task set generated for the processor with 4 cores

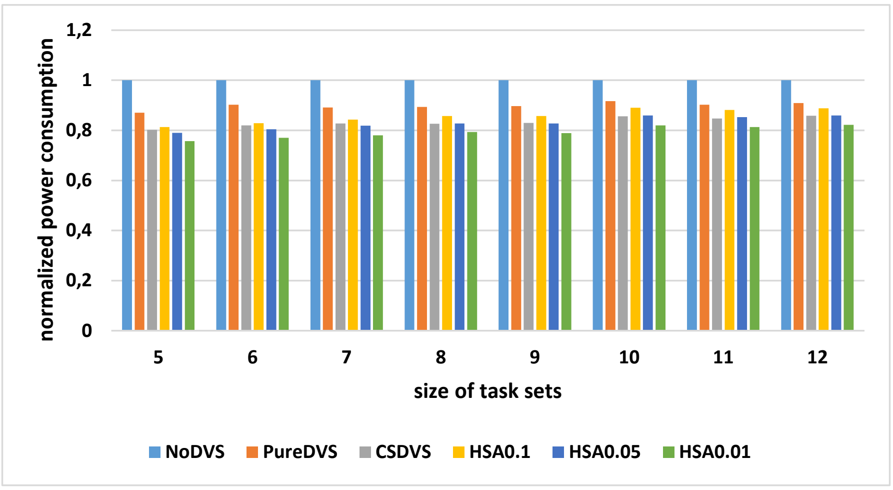


Figure 8.1: Energy reduction efficiency comparison among different algorithms in terms of task number (Intel XScale processor based platform)

contains more tasks and the utilization is higher. Table 8.4 summarizes the properties of generated task sets.

### 8.2.3 Energy Reduction Efficiency (Static Slack)

In this subsection, the proposed HSASC and HSAMC algorithms are compared with existing approaches in terms of energy reduction efficiency in the absence of dynamic slack, i.e., all the tasks run until their WCETs. Before the evaluation results are presented, a set of relevant reference algorithms are first explained.

In the context of the RTSC problem, 3 algorithms from existing studies are selected for comparison:

- NoDVS: There is no DVS strategy at all. All the tasks run at the highest speed. This algorithm is in fact a DPM preferred version, because all the tasks are finished as soon as possible. There is more idle time for the DPM usage. Note that the solution computed by this strategy is also used as the initial solution for the HSASC algorithm.
- PureDVS: This is the counterpart of NoDVS, where all the tasks are executed at the lowest possible speed. This algorithm is a DVS preferred version. Since running all the tasks at the lowest speed may violate hard real-time constraints, i.e., the lowest speed may not always ensure system schedulability, hereby a solution described in Proposition 1 in [Ayd+04] is adopted. In brief, they computed a constant speed $\overline{S}$ for all the tasks within a continuous range of speeds, i.e., they have assumed an ideal DVS model. This work, however, focuses on a more realistic

model, which provides a finite number of discrete operating frequencies. Thus, their solution is adapted by selecting the next available speed above $\overline{S}$.

- CSDVS: This is a critical speed oriented DVS strategy, where all the tasks are executed at the respective critical speed. There is a similar problem here as for PureDVS, because the critical speed is computed without taking into consideration system feasibility. Therefore, the speed assignment scheme from [JG04] is adopted in this case. Basically, all the tasks are initially assigned with their critical speeds. Afterward, a heuristic is developed to incrementally and iteratively increase the assigned speeds until system schedulability is ensured. More specifically, in each iteration all the tasks are investigated and one task will be selected, so that its assigned speed is increased by one level, i.e., to the next higher speed. Concerning task selection, the one resulting in minimal energy increase is selected.

Figure 8.1 shows the simulation results on the Intel XScale processor platform comparing the HSASC algorithm with the other algorithms in terms of energy reduction efficiency. Hereby `HSAx` indicates the HSASC algorithm with the termination parameter $\beta = x$, e.g., `HSA0.01` has the termination parameter $\beta = 0.01$. Moreover, the shown results are classified in terms of number of tasks in a task set, which is indicated by the x-axis. In addition, the y-axis shows solution values, i.e., system energy consumption, by applying different algorithms. Note that hereby all the solution values are normalized with regard to the value obtained by NoDVS.

By looking at the HSASC algorithm, it is obvious that the smaller the $\beta$, the better result the HSASC algorithm can achieve. As expected, `HSA0.01` saves more energy than `HSA0.05` and `HSA0.1`. Furthermore, this experiment shows that `HSA0.01` algorithm outperforms all the other reference algorithms in all cases. More interestingly, the energy reduction efficiency of the HSASC algorithm remains at the same level, as the number of tasks increases. This obviously indicates that the HSASC algorithm scales well in terms of input size.

Similar results can be observed for ARM based platforms as well, which are shown in Figure 8.2.

In the context of RTMC, the LA+LTF+FF algorithm from [CHK06] is chosen as a reference. LA+LTF+FF is a two stage static approach where in the first stage it tries to perform a balanced task partition and in the second stage some tasks are reallocated, so that some processor cores could become idle and therefore may be completely shut down at run-time. More concretely, the first stage adopts the worst fit and largest task first strategy, i.e., the tasks are sorted in a non-increasing order of their utilization at the beginning. The

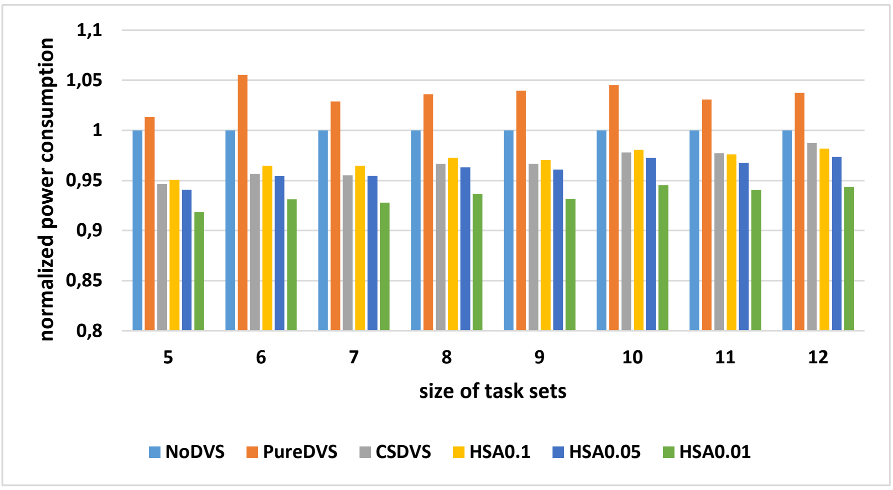


Figure 8.2: Energy reduction efficiency comparison among different algorithms in terms of task number (ARM Cortex-A8 processor based platform)

algorithm then partitions a task, in the task order, to the processor core with the smallest utilization among all the cores, i.e., the most lightly loaded core. After the task partition, for each processor core there will be a frequency selected and then assigned to all the tasks on the core. This frequency is chosen as low as possible, however above the respective critical speed. In the second stage, the set of processor cores with relatively lower utilization are considered, i.e., the cores with the utilization lower than a defined threshold. The main idea is to ”reshuffle” the tasks on such processor cores, so that some cores may become completely idle. Hereby the first fit strategy is adopted for task reallocation. As has been proven by the authors, the LA+LTF+FF algorithm is in fact an 1.667-approximation algorithm for per-core platforms, provided that the processor cores may operate at any frequency within a given range.

According to Chapter 4, the HSAMC algorithm starts with an initial solution generated by partitioning tasks using the worst fit strategy. In this evaluation, however, the algorithm adopts the solution of the LA+LTF+FF algorithm as the starting point. The main reason is due to extremely large solution space of the RTMC problem and starting with a ”good” solution will significantly improve the search performance.

Figures 8.3 and 8.4 show the simulation results on the `(I2)2` and `I2A2` platforms. Hereby the algorithms are denoted in a similar way as for single-core processor platforms, e.g., `HSAMC0.01` denotes the HSAMC algorithm with $\beta = 0.01$. In the figures, the x-axis shows the number of tasks and the y-axis gives the power consumption of output solution normalized to the solution of the LA+LTF+FF algorithm. Clearly, `HSAMC0.01` achieved the best results in all the cases and the smaller the $\beta$, the more power saving can be obtained. The similar observation can be found as well according to further simula-

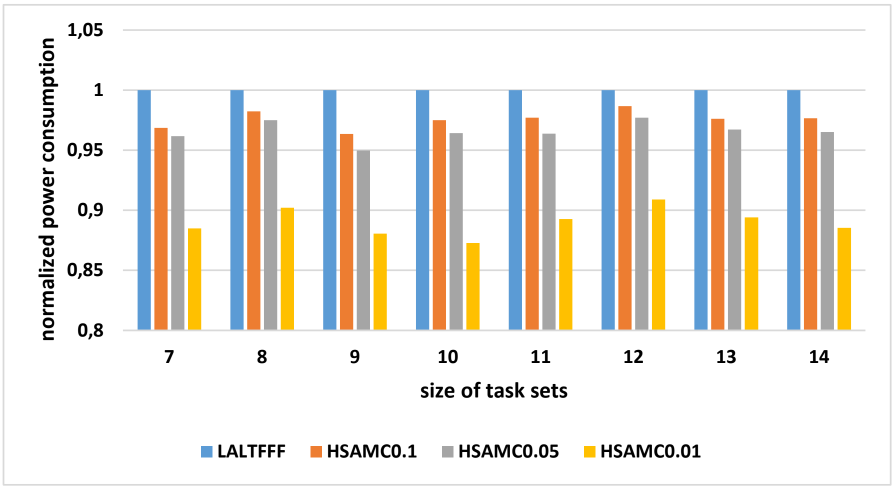


Figure 8.3: Energy reduction efficiency comparison among different algorithms in terms of task number ($(\mathtt{I2})_2$)

tion results on the $\mathtt{I2}$, $(\mathtt{I1})_2$, $\mathtt{I4}$ and $(\mathtt{I1})_4$ platforms, which are provided in Appendix A.1.

### 8.2.4 Energy Reduction Efficiency (Dynamic Slack)

In this section the focus is on energy reduction efficiency taking into consideration task run-time variation, which is the actual execution time of tasks. The test inputs are the same task sets from the previous section. However, run-time variation is allowed and simulated by altering the execution time of each task according to Gaussian distribution [Bry95]. Note that this variation is made in the application stage of the ES-AS approach. For each generated task set, 50 iterations are simulated by using the best solution found in the exploration stage. The average power consumption over 50 iterations is computed. Basically, the results in this subsection provides the energy reduction in application stage, whereas the results from previous subsection presented the energy reduction in exploration stage. In other words, the results here show further energy reduction in addition to the previous results by considering dynamic slack.

To simulate run-time variation, let $W_i$ and $B_i$ denote the worst case and best case execution time of the task $\tau_i$, respectively. $B_i$ is defined by $B_i = W_i \cdot (1 - \gamma)$ with $0 < \gamma < 1$. The Gaussian function is then defined with dependence to $W_i$ and $\gamma$, namely the mean is

$$\mu = \frac{W_i + B_i}{2} = W_i \cdot \frac{2 - \gamma}{2} \tag{8.1}$$

and the variance is

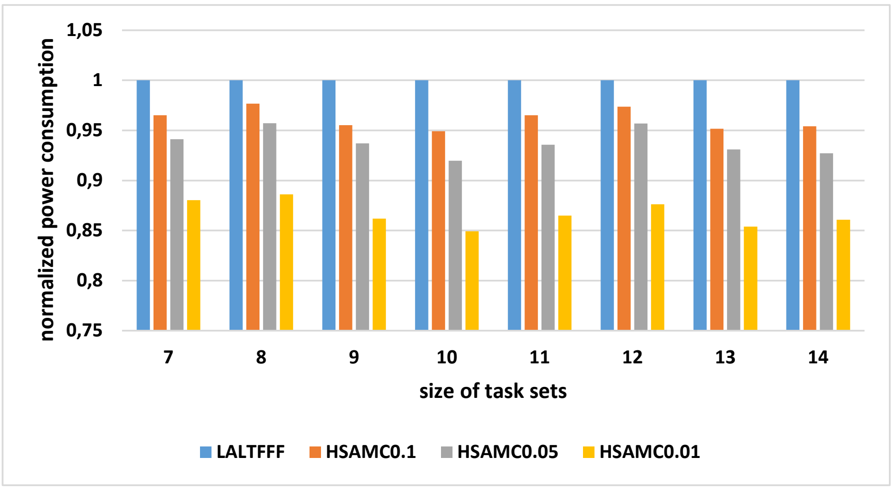


Figure 8.4: Energy reduction efficiency comparison among different algorithms in terms of task number (I2A2)

$$\sigma^2 = 0.4 \cdot \frac{W_i - B_i}{2} = 0.2 \cdot \gamma \cdot W_i \tag{8.2}$$

Obviously, the greater the $\gamma$, the more variation is allowed. In what follows, the simulation results on single-core based platforms are first presented and the results on multi-core processor platforms are discussed afterward.

Based on the Intel XScale processor, Figure 8.5, 8.6 and 8.7 show the evaluation results for the algorithms `HSA0.01`, `HSA0.05` and `HSA0.1`. Again, hereby the results are classified in terms of task number shown by the x-axis. In addition, the y-axis shows the average energy consumption normalized with regard to the respective algorithm without run-time variation. The bars with different colors indicate different $\gamma$ values. As expected, the larger the $\gamma$, the more the actual task execution time varies from the WCET and therefore the more dynamic slack is available. As a result, more power can be saved. Besides, the energy reduction clearly scales well as the number of tasks increases.

Based on the ARM processor, similar simulation results are observed and shown in Figure 8.8, 8.9 and 8.10.

In the context of multi-core processor platforms, Figure 8.11 and 8.12 show the simulation results with regard to the `HSAMC0.01` algorithm on the `(I2)`$_2$ and `I2A2` platforms, respectively. The x-axis shows the number of tasks and the y-axis gives the normalized power consumption, where `HSAMC0.01` is selected as the reference. As expected, the larger the $\gamma$, the more power can be saved, because more dynamic slack can be utilized. Besides, the power reduction is independent of task number. Similar results are as well observed on the `I2`, `(I1)`$_2$, `I4` and `(I1)`$_4$ platforms, which are shown in Appendix A.2.

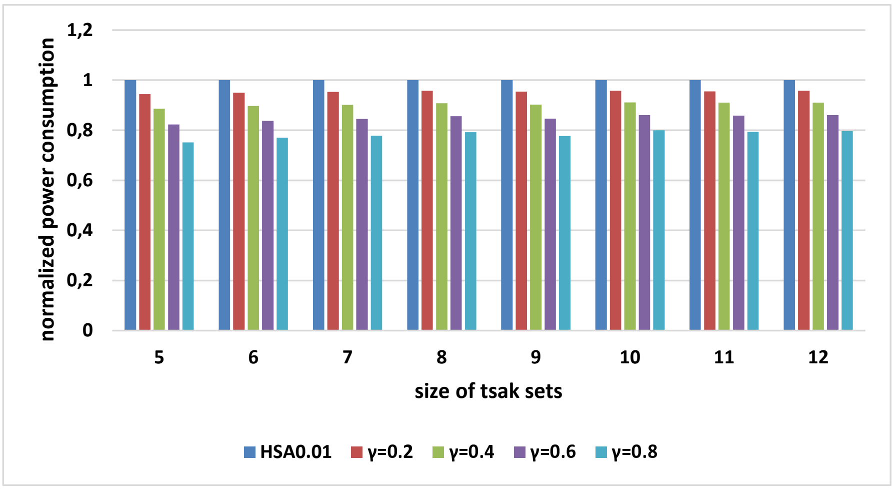


Figure 8.5: Average power consumption comparison using different γ in terms of task number (Intel XScale processor based platform, `HSA0.01`)

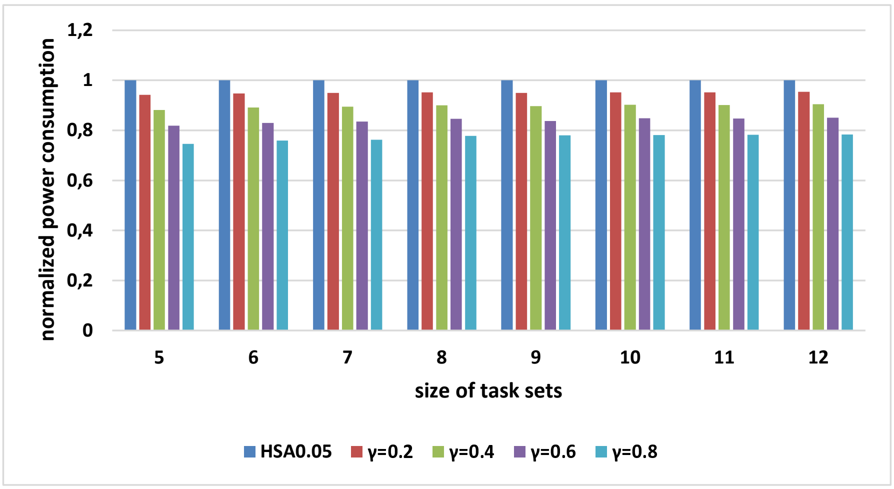


Figure 8.6: Average power consumption comparison using different γ in terms of task number (Intel XScale processor based platform, `HSA0.05`)

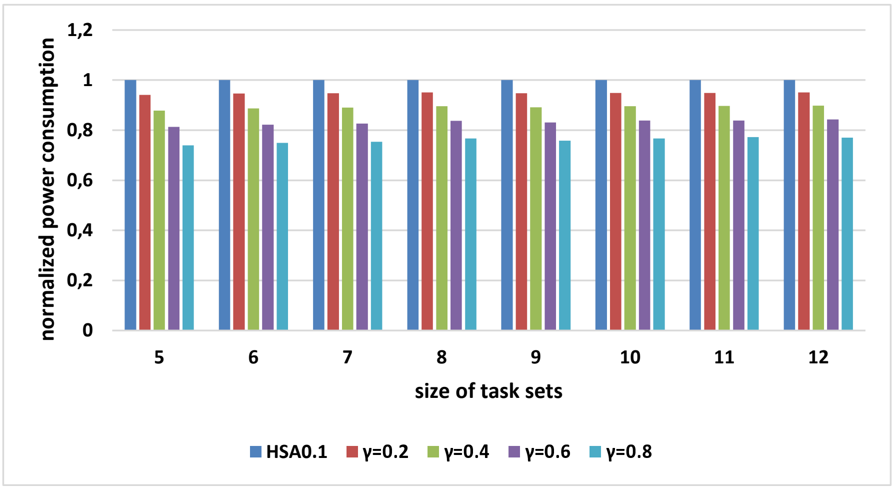


Figure 8.7: Average power consumption comparison using different $\gamma$ in terms of task number (Intel XScale processor based platform, `HSA0.1`)

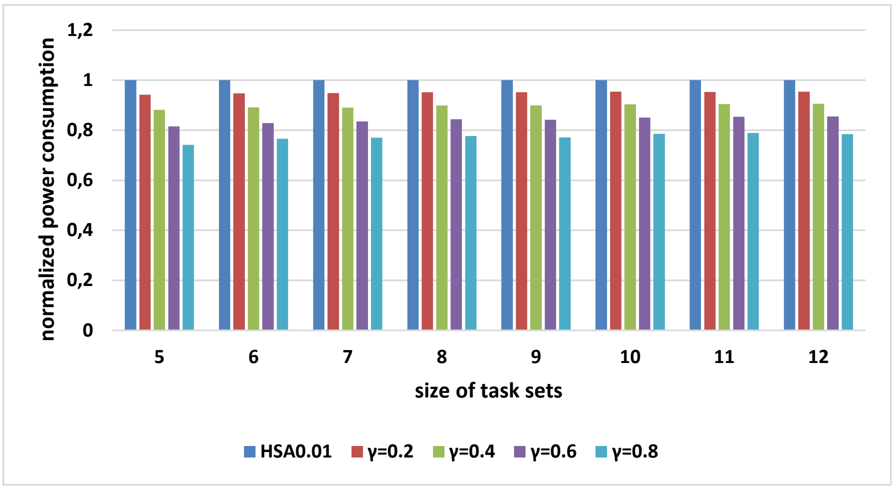


Figure 8.8: Average power consumption comparison using different $\gamma$ in terms of task number (ARM Cortex-A8 processor based platform, `HSA0.01`)

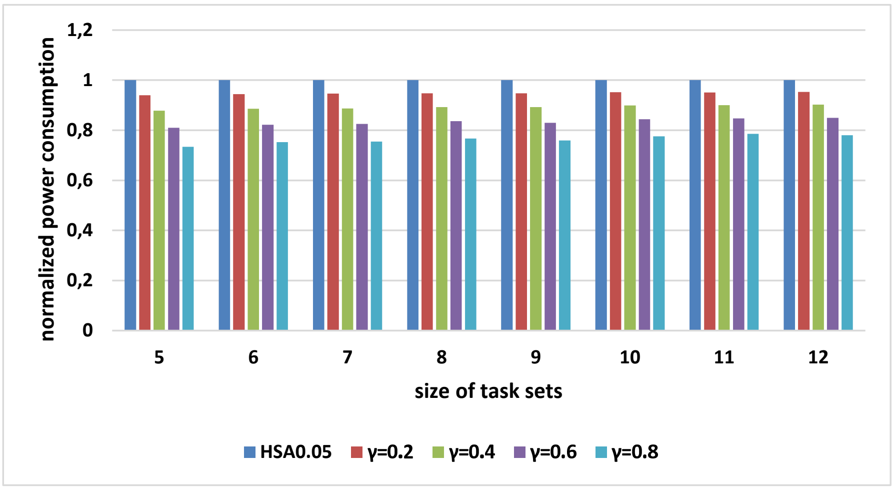


Figure 8.9: Average power consumption comparison using different γ in terms of task number (ARM Cortex-A8 processor based platform, `HSA0.05`)

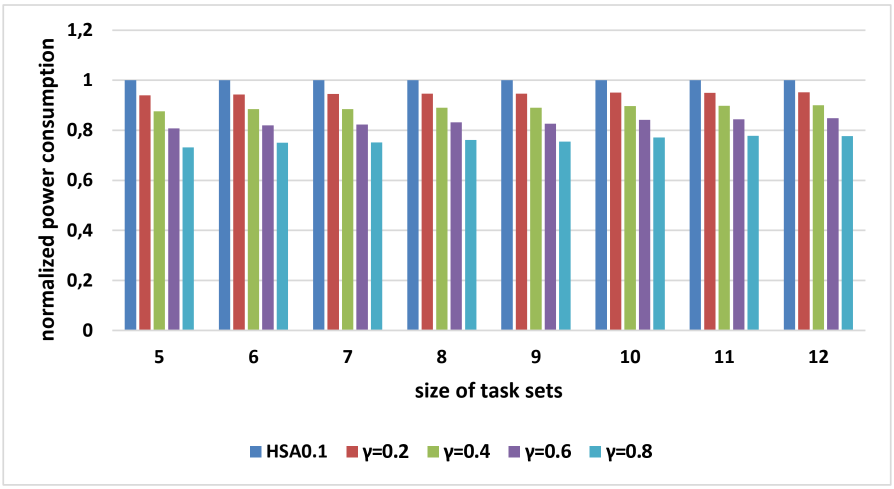


Figure 8.10: Average power consumption comparison using different γ in terms of task number (ARM Cortex-A8 processor based platform, `HSA0.1`)

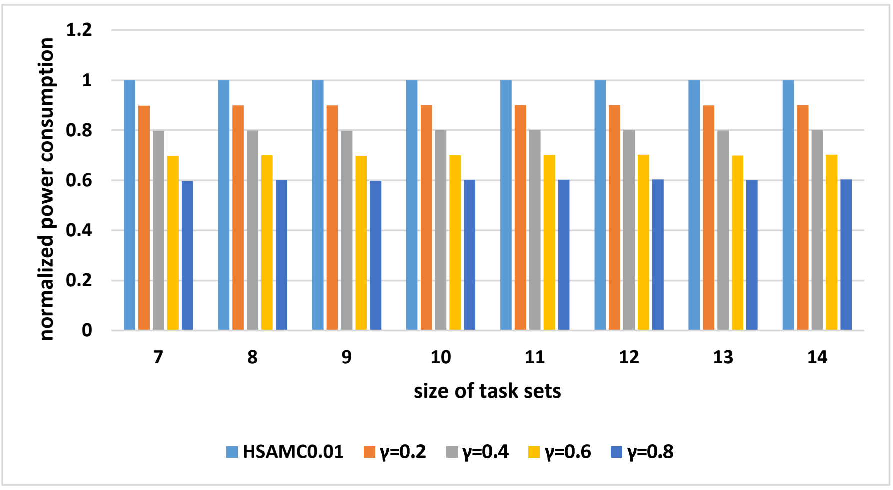


Figure 8.11: Average power consumption comparison using different γ in terms of task number ((I2)2)

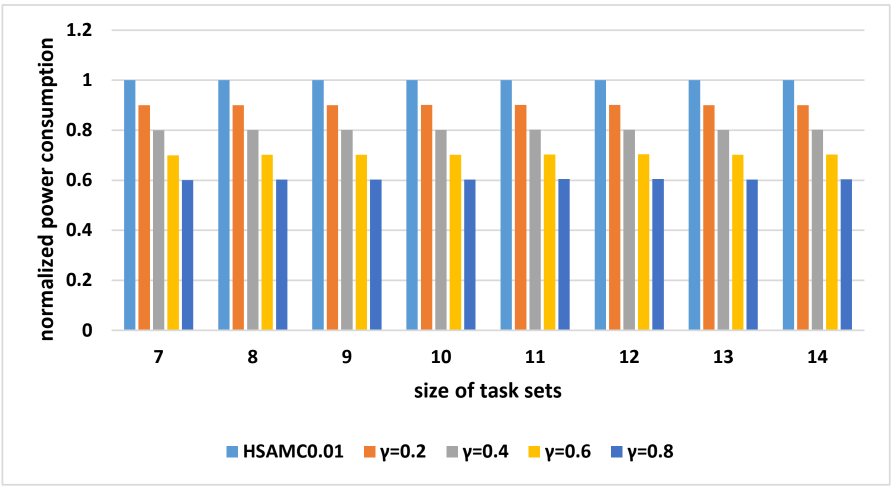


Figure 8.12: Average power consumption comparison using different γ in terms of task number (I2A2)

### 8.2.5 Run-Time Analysis

The previous results show the improvement in terms of energy reduction efficiency by applying the HSASC and HSAMC algorithms. Another important concern is the run-time of these algorithms, i.e., how many iterations they will take to stop according to the proposed termination criterion. Clearly, this is a crucial factor on the duration of exploration stage. Ideally, the exploration stage should be as short as possible. The results shown in this subsection are recorded from the same simulation run in the previous subsections. In what follows, the HSASC algorithm is to be first investigated and the HSAMC algorithm is analyzed afterward.

Figure 8.13 shows the simulation result on Intel XScale processor based platforms by comparing the run-time of different algorithms. The y-axis shows the number of iterations needed by the algorithms while the x-axis shows task numbers. As explained before, `HSAx` indicates the HSASC algorithm with the termination parameter $\beta = x$. `SAx` algorithms are newly introduced, which are presenting the original simulated annealing algorithm without the guideline of neighbor selection, i.e., the neighbors are selected with equal probability. One can clearly observe that `HSA0.01`, `HSA0.05` and `HSA0.1` terminate faster than `SA0.01`, `SA0.05` and `SA0.1`, respectively. More importantly, Figure 8.14 shows that the energy reduction efficiency (shown by the y-axis and normalized with regard to the NoDVS algorithm) obtained by the HSASC and original SA algorithms is comparably equal. This confirms the expectation that the neighbor selection guidelines, which are introduced in the HSASC algorithm, can significantly accelerate termination speed without deteriorating energy reduction efficiency. Another important observation from Figure 8.13 is that the algorithm run-time increases only linearly to the task number, which indicates that the HSASC algorithm scales very well in terms of problem input size. Moreover, if one looks at the case with 12 tasks, the average number of iterations needed by `HSA0.01` is ca. 1100, which is significantly less than the size of the solution space, i.e., $5^{12} = 244140625$ (Note that the number of available speeds in the Intel XScale processor is 5 as shown in Table 8.1).

The similar behavior can be found on ARM Cortex-A8 processor based platforms as well. The simulation results are shown in Figure 8.15 and Figure 8.16. Note that the algorithm run-time hereby is generally faster than the case on Intel XScale processor based platforms, because the ARM processor provides only 3 frequencies (Table 8.2), which is less than the 5 frequencies offered by the Intel XScale processor (Table 8.1). In other words, the solution space here is smaller and therefore a faster termination is expected and reasonable.

In the context of multi-core processor platforms, Figure 8.17 and 8.18 show

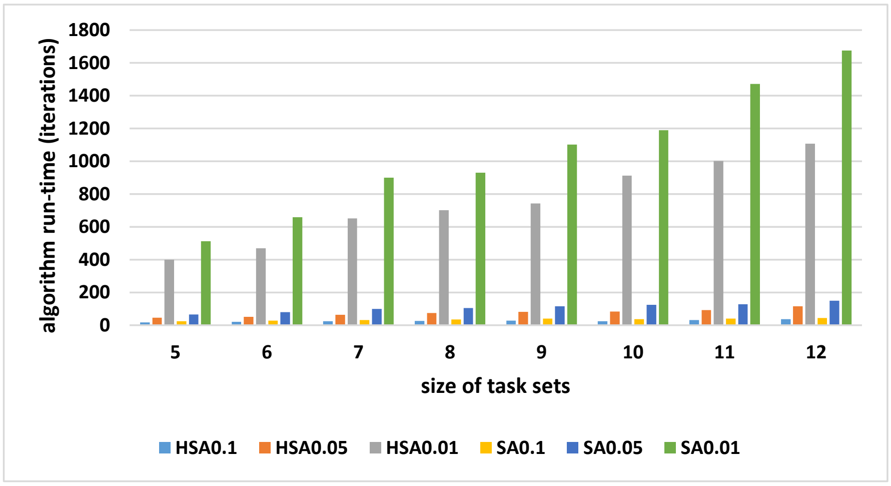


Figure 8.13: Run-time comparison among different algorithms in terms of task number (Intel XScale processor based platform)

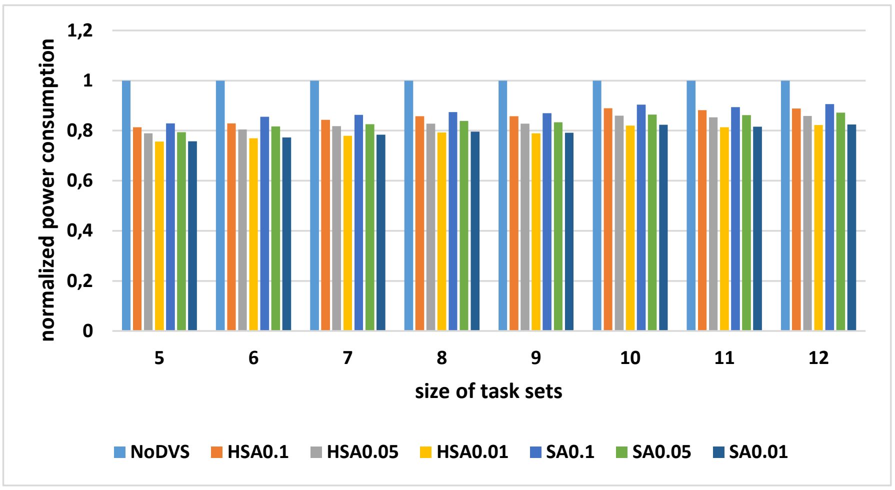


Figure 8.14: Energy reduction efficiency comparison between HSASC and original SA in terms of task number (Intel XScale processor based platform)

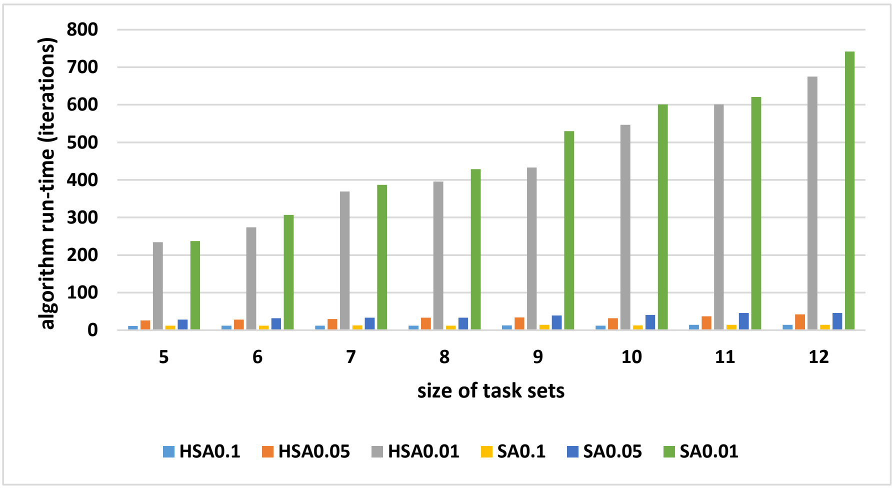


Figure 8.15: Run-time comparison among different algorithms with regard to task number (ARM Cortex-A8 processor based platform)

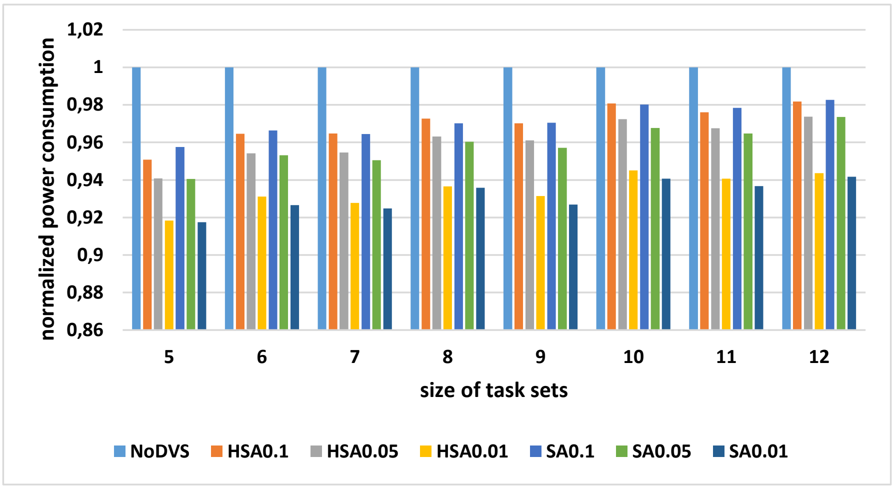


Figure 8.16: Energy reduction efficiency comparison between HSASC and original SA in terms of task number (ARM Cortex-A8 processor based platform)

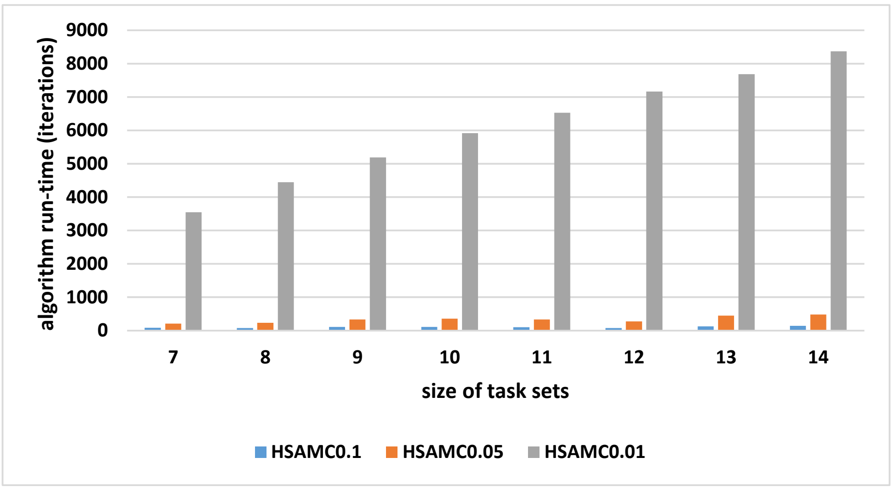


Figure 8.17: Run-time comparison among different algorithms in terms of task number (`(I2)2`)

the simulation results regarding the algorithm run-time on the `(I2)2` and `I2A2` platforms, respectively. The x-axis shows the number of tasks and the y-axis gives the number of iterations required to terminate the exploration stage. This is also the run-time of the HSAMC algorithm. The results clearly confirm the expectation that the smaller the $\beta$, the longer the algorithm takes. Besides, the algorithm run-time increases only linearly in terms of the task number. Similar observation can be as well obtained on the `I2`, `(I1)2`, `I4` and `(I1)4` platforms, which are provided in Appendix A.3.

Furthermore, Figure 8.19 shows the comparison of the `HSAMC0.01` algorithm with the original simulated annealing algorithm, where neighbors are selected according to the uniform distribution. Obviously, `HSAMC0.01` terminates significantly faster than the `SA0.01` algorithm. Hereby the x-axis in the figure shows the different platforms and the y-axis gives the average algorithm run-time over all the generated test cases. More interestingly, though `HSAMC0.01` has a lower run-time, its performance in terms of energy reduction efficiency is even slightly better than `SA0.01` on almost all the platforms. This observation is illustrated in Figure 8.20. Note that hereby the y-axis gives the power consumption of the respective algorithms, which are normalized with regard to the LA+LTF+FF algorithm.

## 8.2.6 Estimation Accuracy

Chapter 5 introduced a regression based technique, which is able to estimate algorithm performance in an online fashion. The main objective of this subsection is to evaluate the accuracy of the algorithm termination criterion. As shown in Chapter 5, the approximation ratio $\varepsilon(\omega)$, defined in (5.13), plays the most important role. It provides the estimated quality of a solution. The

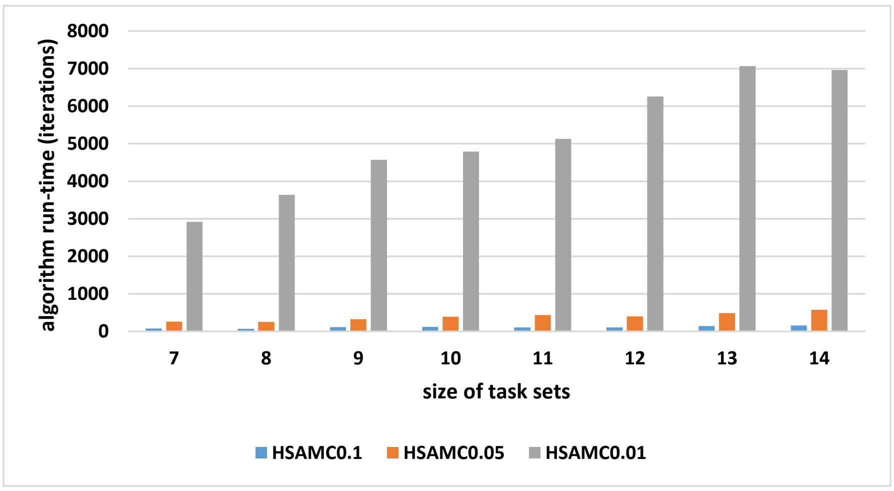


Figure 8.18: Run-time comparison among different algorithms in terms of task number (I2A2)

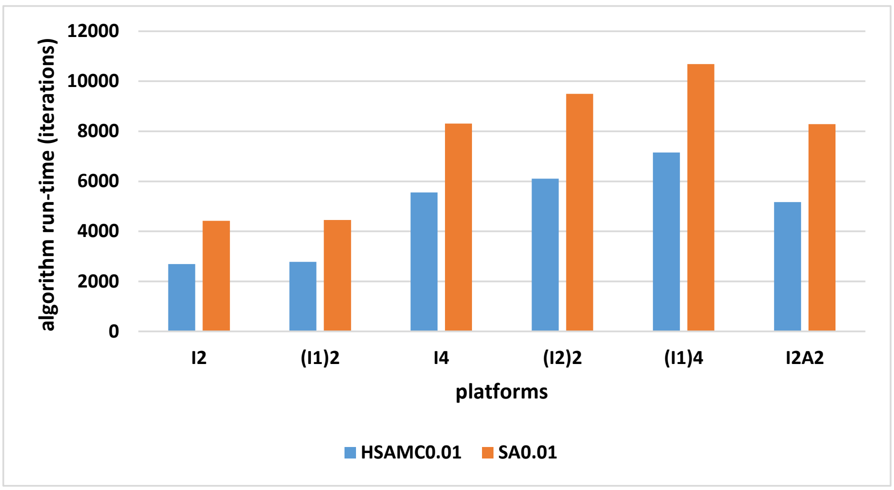


Figure 8.19: Run-time comparison of HSAMC and original SA on different platforms

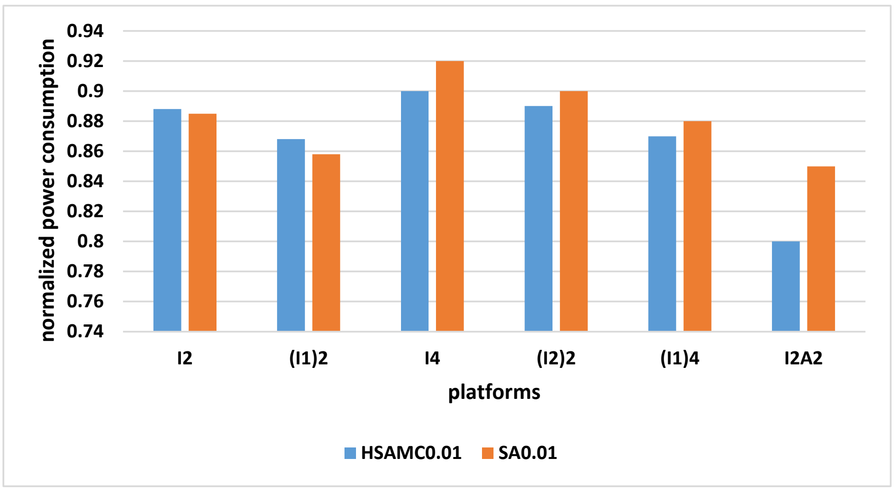


Figure 8.20: Energy reduction efficiency comparison of HSAMC and original SA on different platforms

algorithm terminates, if $\varepsilon(\omega)$ is less or equal to the specified termination parameter $\beta$. For instance, `HSA0.01` shown above describes the algorithm that will terminate when the estimated solution approximation ratio is less than or equal to 0.01. That is, $\beta$ indicates the expected approximation ratio of the output solution when the algorithm terminates. However, the main question is that whether the actual approximation ratio of the solution is indeed less than or equal to $\beta$.

In order to investigate the actual approximation ratio, the value of the optimal solution must be present. Unfortunately, obtaining the real optimum value is extremely difficult. Since the RTSC and RTMC problems admit no closed-form to compute a solution value, i.e., $J(\omega)$, well-known optimization problem solver, such as the integer linear programming solver, may not be applied here. The only remaining option is to use the exhaustive search method, which goes through all the possible solutions in a search space. In other words, all the solutions need to be simulated in the abstract RTOS simulation framework. Due to the extremely large size of solution space, this method is not always realizable in practice. For example, one instance of the RTSC problem, which applies the Intel XScale processor (with 5 DVS states) and contains 11 tasks, has a search space with $5^{11} = 48828125$ solutions. Since the simulation of each solution usually takes several milliseconds, the whole process will take ca. 13 hours. As shown in the previous subsections, 200 instances of the RTSC problem (with 11 tasks) are generated, therefore one needs ca. 108 days to complete the tests, which is clearly not feasible. The situation by the RTMC problem is even worse. As a result, this section only shows the evaluation result for the HSASC algorithm applied on the task sets, where the size is less or equal to 10.

Figure 8.21 and 8.22 illustrate the simulation results on Intel and ARM pro-

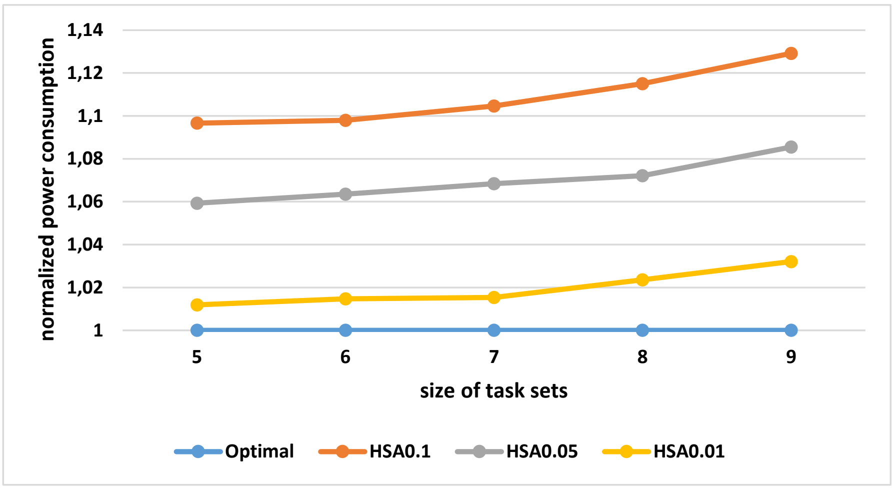


Figure 8.21: Estimation accuracy in terms of task number (Intel XScale processor based platform)

cessor based platforms, respectively. The test input remains the same as in the previous subsections. The y-axis gives the normalized solution value with regard to the optimum value. In fact, the y-axis shows $1+\varepsilon(\omega)$, which implies the actual approximation ratio of a solution. As mentioned before, the optimum value is obtained by using the exhaustive search method in the simulation framework. In addition, the x-axis shows the number of tasks in a task set. This figure confirms that almost all the actual approximation ratios are quite close to the expected ratio $\beta$. One exception is the case with `HSA0.1` on ARM processor based platforms (Figure 8.22). Hereby the main reason is that the approximation ratio of the initial solution (using NoDVS) is already very close to $\beta = 0.1$.

Unfortunately, Figure 8.21 shows a trend that the approximation ratio becomes more inaccurate, as the number of tasks increases. This effect can be eliminated by putting more weight of task number $n$ in the calculation (5.12) of the smoothing constant $\alpha$, which plays a key role to steer the convergence speed. However, this improvement comes at the cost of an increase of algorithm run-time. More discussion on this issue can be found in Chapter 9.

## 8.2.7 ES-AS Run-Time Overhead

In Chapter 4, the run-time overhead introduced by the ES-AS approach for the RTSC and RTMC problems is formally analyzed. However, this theoretical analysis gave only an asymptotic result, i.e., how fast the overhead will increase in terms of input size. There is no information about the constants, which could influence the real overhead as well. Therefore, the main objective of this subsection is to study the actual run-time overhead. Since

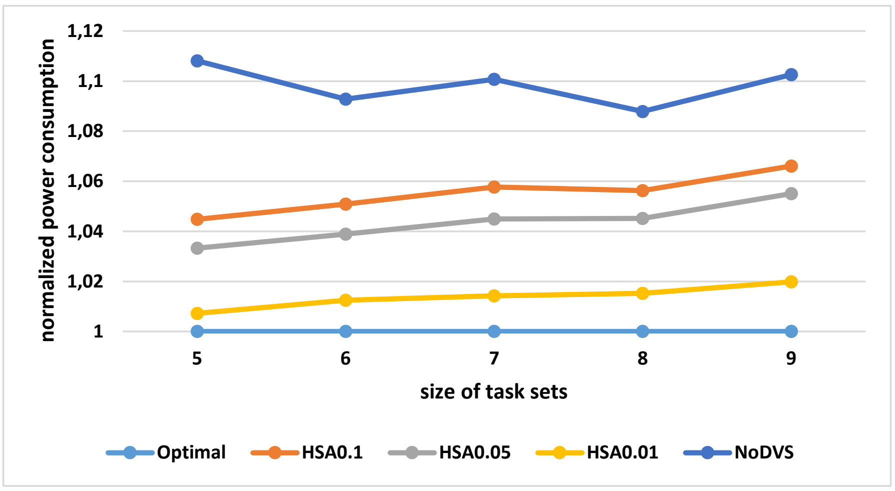


Figure 8.22: Estimation accuracy in terms of task number (ARM Cortex-A8 processor based platform)

the evaluation is made by simulation, the overhead of the ES-AS approach is compared with the scheduling overhead caused by the traditional EDF strategy, which is originally implemented in the abstract RTOS simulation framework. More specifically, two metrics to evaluate the run-time overhead are defined in (8.3) and (8.4).

$$\rho_1 = \frac{\text{ES-AS overhead at each hyper period boundary}}{\text{EDF overhead at each scheduling point}} \tag{8.3}$$

$$\rho_2 = \frac{\text{ES-AS overhead at each scheduling point}}{\text{EDF overhead at each scheduling point}} \tag{8.4}$$

Basically, $\rho_1$, also referred to as HP-overhead, reflects the ratio between the overhead coming from the ES-AS approach at each hyper period boundary and the EDF overhead. Note that at each boundary one iteration of the HSASC or HSAMC algorithm is executed. Thus the overhead is measured as how long one iteration will take in average. $\rho_2$, also referred to as SC-overhead, gives the ratio between the total overhead of the ES-AS approach at each scheduling point and the EDF overhead. In what follows, the overhead for the RTSC problem is first evaluated and afterward the overhead for the RTMC problem is investigated.

Figure 8.23 and 8.24 illustrate the HP- and SC-overhead (y-axis) in terms of task number (x-axis) on Intel and ARM processor based platforms, respectively. One can observe that the HP-overhead is generally higher than the SC-overhead. The main reason is due to Theorem 6.3.2, which shows that the overhead introduced by the ES-AS approach at each hyper period boundary is $O(n)$ being higher than the overhead at each scheduling point, which

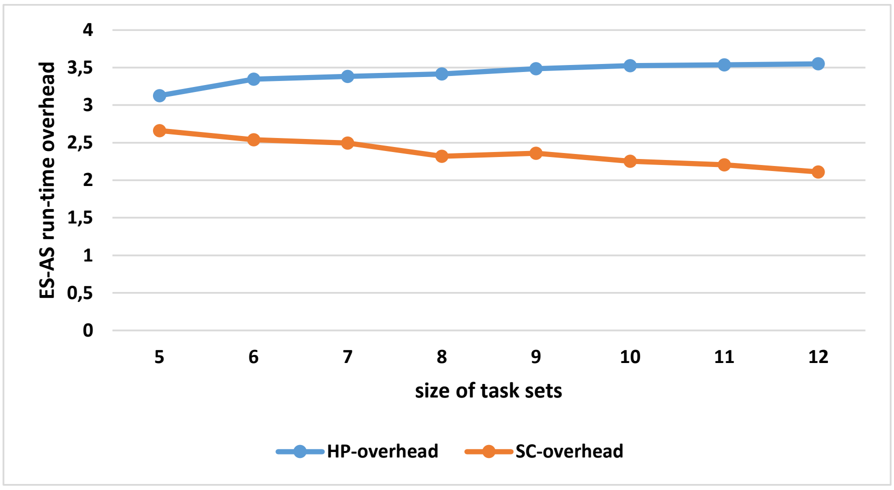


Figure 8.23: ES-AS run-time overhead in terms of task number (Intel XScale processor based platform)

is $O(m \cdot c)$[8]. More interestingly, hereby the HP-overhead increases slightly while the SC-overhead decreases slightly, as the task number increases. This is accounted to the fact that the HP- and SC-overhead are defined as ratios between the actual overhead and the EDF overhead. The actual HP-overhead has a complexity of $O(n)$ and SC-overhead of $O(m \cdot c)$[8] while the time complexity of EDF is $O(\log n)$. In total, it is reasonable to conclude that the actual run-time overhead caused by the ES-AS approach is in the same order of magnitude as the EDF scheduling algorithm.

In case of multi-core processor platforms, the experiments are conducted for the following platforms: `I2`, `(I1)2`, `I4`, `(I2)2` and `(I1)4`. Figure 8.25 shows the HP- and SC-overhead on the `I2` and `(I1)2` platforms in terms of task number. The x-axis shows the number of tasks in a generated task set and the y-axis shows the run-time overhead. In general, the overhead is comparable to the case on single-core platforms, which indicates that the overhead introduced by the ES-AS approach for RTMC is in the same order of magnitude as the EDF scheduling overhead as well. Furthermore, the HP-overhead increases as the number of tasks increases. This is due to the fact that the actual overhead at each hyper period boundary is $O(n)$ and the EDF overhead is $O(\log n)$. Moreover, since the ES-AS overhead for multi-core processor platforms at each scheduling point is $O(\log n + c)$[9] according to Theorem 6.4.1, the SC-overhead keeps constant as the number of tasks increases.

In addition, if one looks at Figure 8.25 and focuses on the results obtained for task sets with the same size, it is clear that both the HP- and SC-overhead decrease as the number of clusters increases, i.e., the cluster size decreases.

[8]The overhead hereby is $O(m \cdot c)$ according to Theorem 6.3.2. Since the number of devices $m$ and the number of supported low power states $c$ are relatively small and constant, the overhead can be considered as $O(1)$ in practice.

[9]If the number of low power states $c$ is considered as a constant, this overhead becomes $O(\log n)$.

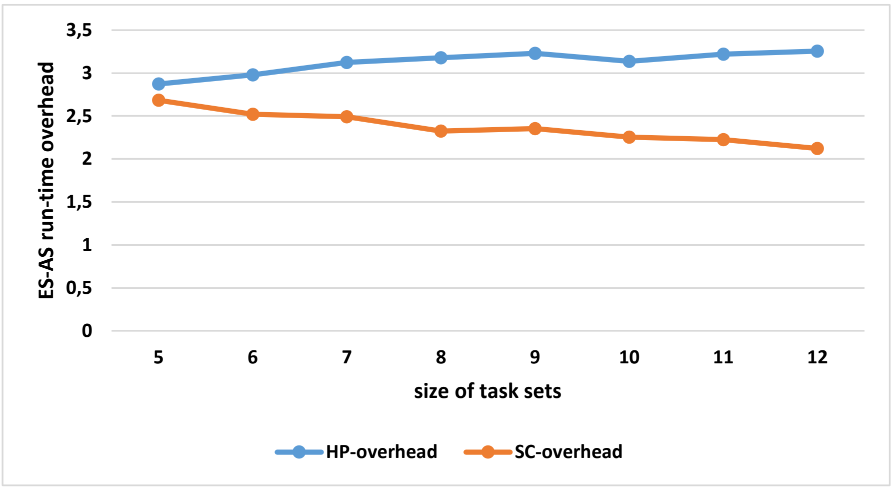


Figure 8.24: ES-AS run-time overhead in terms of task number (ARM Cortex-A8 processor based platform)

For instance, in case of task sets with 7 or 8 tasks, the HP- and SC-Overhead on `I2` are higher than them on `(I1)`$_2$, respectively. Through inspection of the simulation results, the main reason is due to the fact that the EDF overhead decreases as the cluster size increases while the actual HP- and SC-overhead remains. The main reason in turn for the EDF overhead decrease is that a processor with larger cluster size incurs more scheduling points. A processor core $O_i$ may need to change its operating speed, if a scheduling point on another core $O_j$ in the same cluster occurs, even though there is no task event on $O_i$. In such case the EDF scheduling overhead on $O_i$ is rather low, because there is no need to change the running task but only the speed. Since the final result is a mean value over all the scheduling points, a multi-core processor with larger cluster size yields a lower overhead in average.

Moreover, Figure 8.26 shows the simulation results on multi-core processors with 4 cores. As shown in the x-axis, hereby the range of considered task number is different to them for dual-core processors (cf. Table 8.4). Generally, the same trend as described previously can be observed as well.

### 8.2.8 Impact of CP and SIP

This subsection investigates the impact of the conservative protocol and the speed inheritance protocol, which are described in Chapter 7. Since they are proposed for multi-core processor platforms, especially for full-chip and cluster-based multi-core processors, hereby the simulation is focused on two platforms: `I4` and `(I2)`$_2$. In total, two experiments are made in this evaluation work.

In the first experiment the main goal is to show the necessity of handling

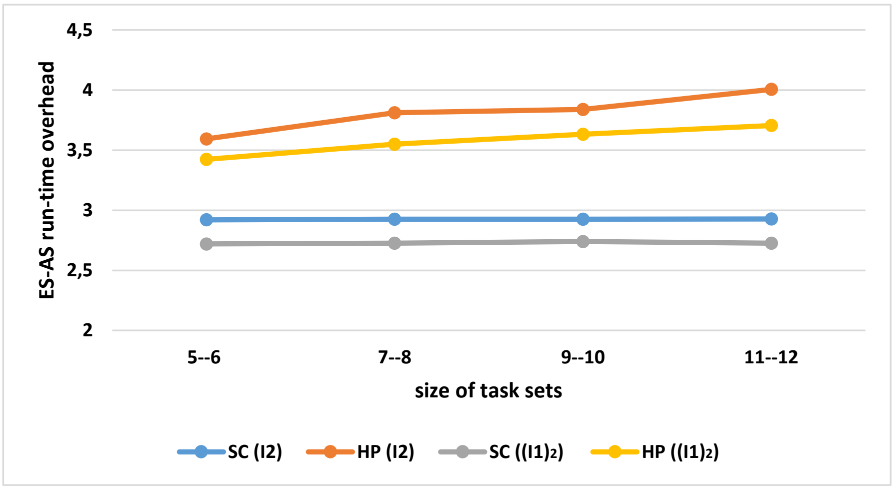


Figure 8.25: ES-AS run-time overhead in terms of task number ( `I2` and `(I1)`$_2$)

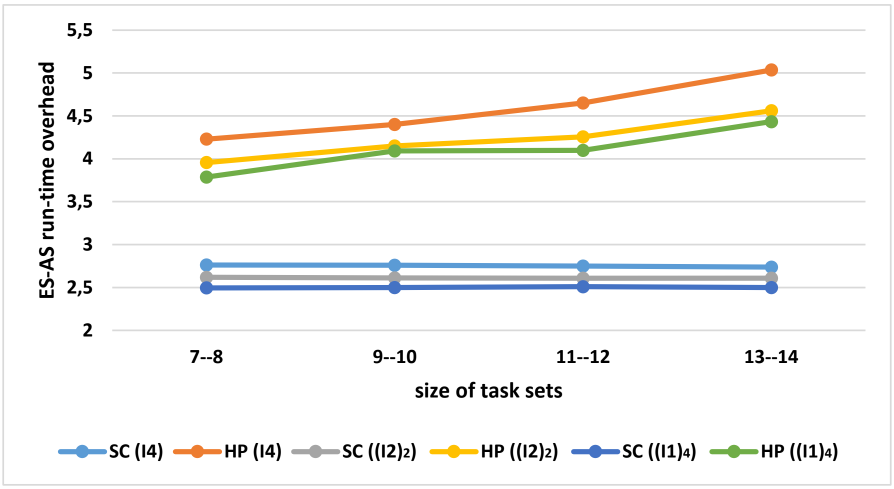


Figure 8.26: ES-AS run-time overhead in terms of task number (`I4`, `(I2)`$_2$ and `(I1)`$_4$)

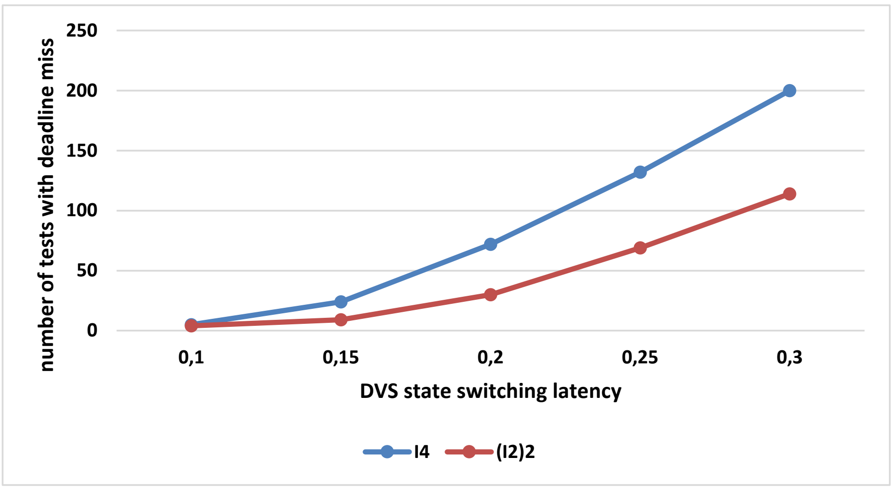


Figure 8.27: Number of tests with deadline miss when DVS state switching overhead is not handled

DVS state switching latency. The simulation runs each generated task set (cf. Table 8.4) for one hyper period using the solution computed by the LA+LTF+FF algorithm. Hereby difference in DVS state switching latency is investigated. Figure 8.27 shows the result, where DVS state switching latency is not treated. The x-axis shows different settings of DVS state switching latency, which is normalized with regard to the average execution time of all the tasks in the generated task sets. The y-axis shows the number of tests, where a deadline miss occurs. This figure clearly confirms the expectation that deadline can not be always ensured, if non-negligible DVS state switching latency is not handled appropriately. It is obvious that the higher the overhead, the more cases with deadline miss exist. Besides, one can observe that there are generally more cases with deadline miss on `I4` than `(I2)2`. This is rather plausible, because `I4` has a larger cluster size and there is more potential for ICPs (Definition 7.3.1). Note that ICP is the main cause for delays and thus also for deadline misses.

Furthermore, the second experiment is to investigate the difference between CP and SIP. Since both protocols provide only sufficient schedulability test, hereby the main focus is to compare the overestimation. That is, how many tests are predicted as non-schedulable, but they are indeed feasible. Formally, it is defined as false negative errors shown below.

**Definition 8.2.1.** *A **false negative error** in the context of CP and SIP is a task set that fails the schedulability test, however, the task set is indeed schedulable using the respective protocol.*

Based on this definition, the false negative error rate of CP ($\rho_{CP}$) and SIP ($\rho_{SIP}$) are introduced as follows.

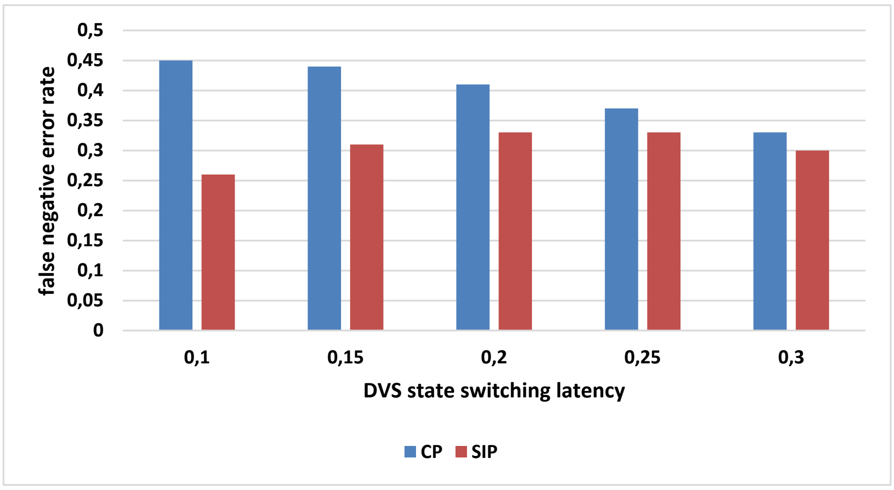


Figure 8.28: The comparison of false negative error rate of CP and SIP on `I4`

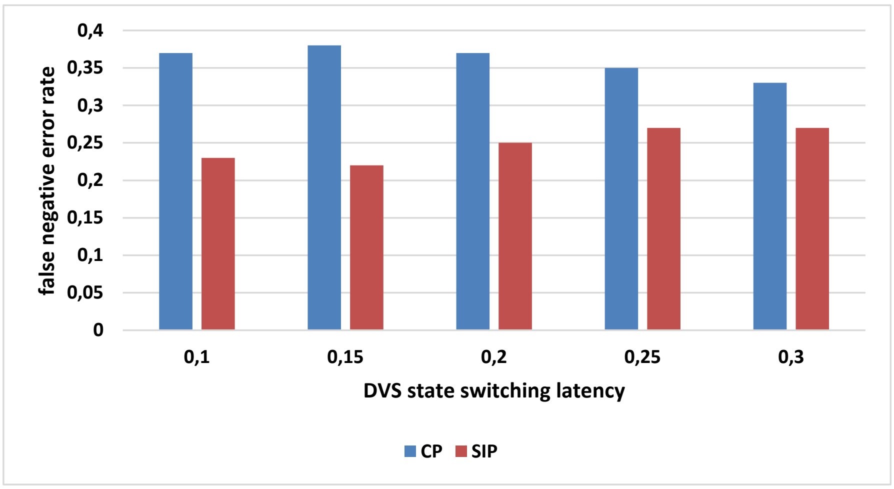


Figure 8.29: The comparison of false negative error rate of CP and SIP on `(I2)2`

$$\rho_{CP} = \frac{\text{number of false negative errors using CP}}{\text{total number of tests}} \tag{8.5}$$

$$\rho_{SIP} = \frac{\text{number of false negative errors using SIP}}{\text{total number of tests}} \tag{8.6}$$

In this case, 1600 task sets (cf. Table 8.4) are simulated by applying CP and SIP, respectively. Figure 8.28 and 8.29 show the evaluation results on the `I4` and `(I2)2` platforms, respectively. In both figures, the x-axis shows the different settings of DVS state switching latency while the y-axis gives the respective false negative error rate. The result confirms the expectation that SIP yields lower error rate than CP on all the platforms.

## 8.3 Real-Life Case Studies

In order to further demonstrate the applicability of the proposed algorithms and approaches, this section presents two experiments implementing real-life case studies on real target platforms. More concretely, the first subsection concentrates on a single-core processor platform, where the BeagleBoard [Beaa] produced by Texas Instruments [Inse] is applied. Afterward, the subsequent subsection shows an evaluation addressing multi-core processor platforms, where the Intel® Core™2 Duo [Intd] processor is used.

### 8.3.1 Single-Core Processor Platform: BeagleBoard

This subsection investigates the applicability of the proposed algorithms and approaches by means of the BeagleBoard, which is a low-cost single-board computer produced by Texas Instruments. Before the evaluation results are discussed, a short introduction of the BeagleBoard, specially with focus on the power model of the processor and peripheral devices, is given. Besides, the energy aware real-time scheduling in this experiment is implemented on the basis of an open source real-time operating system, Organic Reconfigurable Operating System (ORCOS) [ORC]. Thus a brief review of ORCOS is presented as well.

In general, the BeagleBoard provides functionality of a basic computer and is designed primarily with the focus on open source software development. The core component in the BeagleBoard is the OMAP3530 application processor [Insc], which is an ARM Cortex-A8 based System-on-a-Chip (SoC) from Texas Instruments using 65 nm process technology. The OMAP3530 is mainly composed of several subsystems shown as follows [Insd].

- MPU subsystem: mainly it contains an ARM Cortex-A8 processor and a NEON SIMD coprocessor.
- IVA2.2 subsystem: it mainly includes a video and audio accelerator based on the Texas Instruments TMS320DMC64x+ VLIW DSP core.
- On-chip memory: 112 KB ROM and 64 KB SRAM.
- External memory interfaces: general purpose memory controller and SDRAM controller.
- DMA controller: it allows memory-to-memory, memory-to-peripheral and peripheral-to-memory transfer.
- Graphic accelerator subsystem: it mainly contains a 3D graphics accelerator (SGX) to enable high-performance graphic processing.

| State | $F$ | $P$ (mW) | $T_{on\to off}$ | $T_{off\to on}$ | $P_{on\to off}$ | $P_{off\to on}$ |
|---|---|---|---|---|---|---|
| $S_1$ | 1[10] | 999.9 | n.a.[11] | n.a. | n.a. | n.a. |
| $S_2$ | 0.7 | 548.64 | n.a. | n.a. | n.a. | n.a. |
| $S_3$ | 0.17 | 90.55 | n.a. | n.a. | n.a. | n.a. |
| $C_1$ | n.a. | 1.94 | 163 µs | 163 µs | 60 mW | 60 mW |
| $D_0$[12] | n.a. | 700 | n.a. | n.a. | n.a. | n.a. |
| $D_1$ | n.a. | 0 | 50 ms | 50 ms | 50 mW | 50 mW |

Table 8.5: The power model of the ARM processor and the display subsystem in the BeagleBoard

- Multimedia accelerators: camera subsystem for image capture and display subsystem for image output to an external monitor.
- Power management subsystem: in particular it enables DPM and DVS for processors and peripheral devices.
- Diverse peripherals: for instance, USB host controller, UART controller, I$^2$C controller, etc..

In order to apply these features, the BeagleBoard provides external connectors that can be used to access them [Beab]. For instance, a DVI-D and an S-Video connector are offered to connect external monitors with the display subsystem. A USB host port is provided for serial communication between external devices and the USB host controller in OMAP3530. Because the BeagleBoard is particularly constructed to evaluate the basic capability of OMAP3530 and is not indented as a full development board, not all the interfaces and controllers in OMAP3530 are accessible.

One main reason why the BeagleBoard is selected here is that it provides fairly advanced power management possibilities. The previously mentioned subsystems are managed in separated power domains and thus can be shutdown independently. In the context of this evaluation, the main components under consideration are the ARM Cortex-A8 processor and the display subsystem. The power model of the ARM processor is taken from Table 8.2. For the display subsystem, the power consumption of the entire board is measured using a digital energy meter [AG]. Hereby one measures the case where the display subsystem is activated and the case where it is disabled. Thus the difference of the two measurements gives the power consumption of the display subsystem. As a summary, Table 8.5 lists the power model of the considered components.

[10]The operating frequency without normalization is 600 MHz.

[11]not applicable.

[12]Power states of the display subsystem.

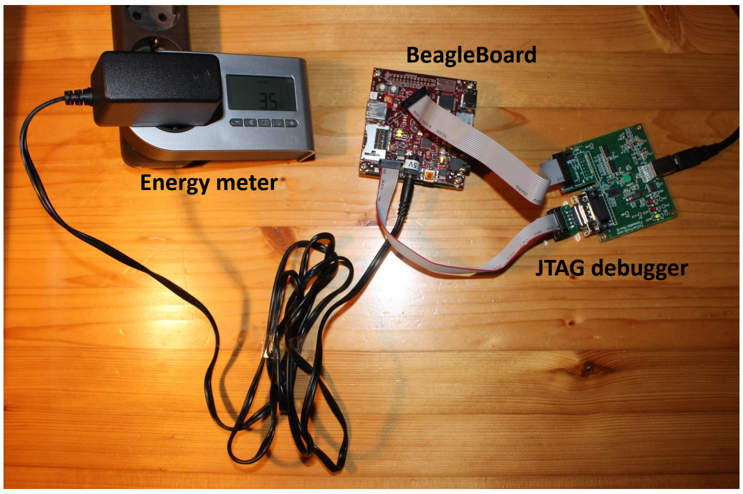


Figure 8.30: The experiment setup with BeagleBoard

| Task | WCET (ms)[13] | Period/Deadline (ms) | *Dev* |
|---|---|---|---|
| GUI control ($\tau_1$) | 2.5 | 100 | $\varnothing$ |
| Image processing ($\tau_2$) | 50 | 500 | $\varnothing$ |
| Visualization ($\tau_3$) | 25 | 500 | $\{R_1\}$[14] |
| EC ($\tau_4$) | 12.5 | 1000 | $\varnothing$ |
| Servo control[15] ($\tau_5$) | 10 | 100 | $\varnothing$ |
| Sensor control[16] ($\tau_6$) | 5 | 100 | $\varnothing$ |

Table 8.6: Case study: X-Ray machine [Gro10]

In order to debug and program the OMAP3530 chip, a compact USB to JTAG in-circuit debugger/programmer called Flyswatter [Tooa] from Tin Can Tools [Toob] together with the OpenOCD (version 0.7.0) [Ope] debugging software are used. In total, the experiment setup is illustrated in Figure 8.30.

In order to implement an energy aware real-time scheduling, an RTOS is needed. Hereby ORCOS [ORC], which is an open-source project developed at the University of Paderborn, is applied. The main property of ORCOS is that it has a fairly small footprint and is highly configurable. Currently, ORCOS supports PowerPC, SPARC and ARM architectures. The EDF and RM real-time scheduling algorithms are already implemented in ORCOS. In this evaluation work, ORCOS is extended in two aspects. First, the DPM and DVS techniques for the processor and devices are implemented. Second, the proposed ES-AS approach is implemented in ORCOS.

[13]Running at 600 MHz.

[14]$R_1$ denotes the display subsystem.

[15]To simplify the implementation, the two servo tasks with the same period [Gro10] are combined into one task.

[16]To simplify the implementation, the two sensor tasks with the same period [Gro10] are combined into one task.

| Task | $\tau_1$ | $\tau_2$ | $\tau_3$ | $\tau_4$ | $\tau_5$ | $\tau_6$ |
|---|---|---|---|---|---|---|
| Speed assignment | $S_3$ | $S_2$ | $S_2$ | $S_2$ | $S_3$ | $S_2$ |

Table 8.7: Computed speed assignment of the tasks

For the application, this experiment selects one case study from [Gro10], i.e., the X-ray machine. This scenario shows a typical application in the medical domain, which mainly controls the actuators to position the X-ray beam properly. This application is composed of two subsystems, where the first subsystem is responsible for user interface and the second one is the control software of the X-ray unit. In brief, the user interface subsystem contains three tasks including GUI control, image processing and visualization. Hereby one assumes that the visualization task needs the display subsystem of the BeagleBoard. The X-ray control software contains sensor tasks, actuator controlling tasks and an automatic exposure control (EC), which performs the X-ray termination if it is not done by a human operator. Table 8.6 gives a summary of the tasks in this case study.

As the result by using the `HSA0.01` algorithm, the system needs 341 hyper periods to terminate the exploration stage and the computed speed assignment is shown in Table 8.7. The application has a hyper period of 1 s. Therefore the exploration stage takes 341 s in total. By reading the measured average power consumption of the BeagleBoard from the energy meter, the computed solution consumes 2.6 W, whereas running all the tasks at the highest speed consumes 2.9 W. This indicates a 10% power reduction by using the proposed HSASC algorithm.

### 8.3.2 Multi-Core Processor Platform: Intel Core 2 Duo Processor

This subsection focuses on the evaluation of real-life case studies on multi-core processor platforms. For this, the Lenovo ThinkPad T400 notebook [Len] is selected as the hardware platform. The main component involved in this experiment is the built-in Intel Core 2 Duo P9500 processor [Intb], which in total supports three operating frequencies: 2533 MHz, 1600 MHz and 800 MHz. This processor presents a full-chip platform, which means that all the cores may only operate at the same speed at the same time. Based on the data sheet [Inta], unfortunately, the power consumption of the three active states can not be directly acquired. However, its power consumption in low power state (`Deeper Sleep`) is available and equal to 4.75 W. In order to obtain the complete power model, the power consumption of the entire notebook is measured using the aforementioned energy meter. In the measurement, all the unneeded devices or services (e.g., bluetooth, wireless lan, off-chip graphic card, etc.) are deactivated and the battery is removed. As a result, Table 8.8 shows the average power consumption measured by running the

| Frequency (MHz) | 2533 | 1600 | 800 |
|---|---|---|---|
| $P^l_{t400}$ (W) | 42.5 | 31.8 | 23.4 |
| $P_{t400}$ (W) | 17.4 | 17.4 | 17.4 |

Table 8.8: Power consumption measured for the Lenovo T400 notebook

”stress” program [Man] (denoted by $P^l_{t400}$) as well as keeping system idle (denoted by $P_{t400}$) under different operating frequencies, respectively. Note that the ”stress” program was used only with the option ”–cpu”, which means that only the processor is stressed.

Clearly hereby the measured result contains not only power consumption of the processor, but also of other components in the notebook that can not be deactivated. To simplify the explanation, $P_{t400}$, $P_{cpu}$ and $P_{other}$ denote the power consumption of the notebook, the processor and the other components, respectively, when the notebook is idle. Conversely, if the notebook is stressed, $P^l_{t400}$, $P^l_{cpu}$ and $P^l_{other}$ denote the power consumption of the notebook, the processor and the other components, respectively. It is obvious that the following equations hold.

$$P_{t400} = P_{cpu} + P_{other} \tag{8.7}$$

$$P^l_{t400} = P^l_{cpu} + P^l_{other} \tag{8.8}$$

$P_{cpu}$ is obviously the processor power consumption in low power state, which is 4.75 W as given previously. According to Table 8.8, $P_{t400}$ is 17.4 W and is independent of operating speed. The reason is rather obvious, because in this case the processor is idle and thus stays in a low power state. As a result, $P_{other}$ can be approximated as follows.

$$\begin{aligned} P_{other} &= P_{t400} - P_{cpu} \\ &= 17.4\text{ W} - 4.75\text{ W} = 12.65\text{ W} \end{aligned} \tag{8.9}$$

As mentioned above, the ”stress” program generates only load on the processor, it is reasonable to assume that $P_{other}$ remains constant regardless whether the notebook is loaded or not, i.e., $P_{other}$ is equal to $P^l_{other}$. Thus, the processor power consumption by running tasks at different operating frequencies can be derived as follows.

$$P^l_{cpu} = P^l_{t400} - P_{other} \tag{8.10}$$

| Frequency (MHz) | 2533 | 1600 | 800 |
|---|---|---|---|
| $P^l_{cpu}$ (W) | 29.85 | 19.15 | 10.8 |

Table 8.9: Power consumption of the processor operating at different speeds

| State | $F$ | $P$ (W) | $T_{on\rightarrow off}$ | $T_{off\rightarrow on}$ | $P_{on\rightarrow off}$ | $P_{off\rightarrow on}$ | $T_{be}$ |
|---|---|---|---|---|---|---|---|
| $S^1_1$ | 1 | 15 | n.a. | n.a. | n.a. | n.a. | n.a. |
| $S^1_2$ | 0.64 | 9.58 | n.a. | n.a. | n.a. | n.a. | n.a. |
| $S^1_3$ | 0.32 | 5.4 | n.a. | n.a. | n.a. | n.a. | n.a. |
| $C^1_1$ | n.a. | 2.3 | 0.01 ms | 0.01 ms | 3 W | 3 W | 0.02 ms |
| $S^2_1$ | 1 | 15 | n.a. | n.a. | n.a. | n.a. | n.a. |
| $S^2_2$ | 0.64 | 9.58 | n.a. | n.a. | n.a. | n.a. | n.a. |
| $S^2_3$ | 0.32 | 5.4 | n.a. | n.a. | n.a. | n.a. | n.a. |
| $C^2_1$ | n.a. | 2.3 | 0.01 ms | 0.01 ms | 3 W | 3 W | 0.02 ms |

Table 8.10: The power model of the Intel Core 2 Duo P9500 processor

For instance, $P^l_{t400}$ by operating at 800 MHz is 23.4 W according to Table 8.8. By subtracting $P_{other}$, the power consumption of the processor ($P^l_{cpu}$) running at 800 MHz is approximately 10.8 W. Table 8.9 gives an overview of the derived power consumption of the processor.

Note that Table 8.9 shows the power consumption of the entire processor, which contains two cores. Thus by further dividing the power consumption, Table 8.10 summaries the complete power model of the Intel Core 2 Duo P9500 processor.

Furthermore, as the real-time operating system this experiment uses RTLinux [THK], which is a patched Linux kernel with capability of real-time computing. More specifically, the RT-PREEMPT patch is able to transform Linux into a fully preemptable kernel. Some of the main features of RTLinux are mentioned below.

- In-kernel locking primitives (e.g., spinlock) are preemptible.
- Interrupt handlers are becoming preemptible.
- User space real-time tasks can be created with specific real-time priority.
- Priority based real-time scheduling is supported.
- User space applications can still use the traditional Linux API.

In this experiment, the Linux kernel 3.4.19 with the RT-PREEMPT patch 3.4.19-rt30 and the RM scheduling algorithm is applied.

| Task | WCET (ms)[17] | Period/Deadline (ms) |
|---|---|---|
| Detection task ($\tau_1$) | 1.5 | 15 |
| Sensor task 1 [18] ($\tau_2$) | 1.8 | 15 |
| Sensor task 2 [19] ($\tau_3$) | 1.2 | 15 |
| Brake pressure control ($\tau_4$) | 20 | 100 |
| Sensor task ($\tau_5$) | 10 | 50 |
| CAN communication ($\tau_6$) | 1 | 5 |

Table 8.11: Case study: air bag control and anti-lock braking system [Gro10]

As the case study, an application in automotive domain is taken from [Gro10]. The original application is composed of three subsystems: an Airbag Control Unit (ACU), an Anti-lock Braking System (ABS) and an Electric Stability Control (ESC). These control systems are commonly referred to as Advanced Driver Assistance Systems (ADAS). Since the total utilization of all the subsystems exceeds the capability of Intel Core 2 Duo P9500, the ACU and ABS are selected for evaluation in this subsection.

The ACU contains one detection task and 8 sensor tasks. The detection task is responsible for monitoring the output of the sensor tasks and inflating the cushion during a collision. To simplify the implementation, this experiment combines the sensor tasks with the same WCET and period into one task. The ABS mainly contains three tasks: the first task is called brake pressure control task, which computes the proper pressure applied for the target actuator. The input of the brake pressure control task is coming from the sensor task (the second task), which provides the rational speed of wheels. The third task is in charge of communication with other systems in car through Controller Area Network (CAN). A summary of the tasks is specified in Table 8.11.

As the result by using the `HSAMC0.01` algorithm, the system needs 200 iterations to terminate the exploration stage. The computed solution is illustrated in Table 8.12. The application has a hyper period of 1.5 s, thus the exploration stage takes 5 minutes in total. By reading the average power consumption of the notebook from the energy meter, the computed solution consumes 26 W whereas the solution of the LA+LTF+FF algorithm consumes 26.5 W. If only the processor power consumption is considered, then power consumption of the `HSAMC0.01` algorithm is 13.35 W and the LA+LTF+FF algorithm consumes 13.85 W. This indicates a 3.6% improvement in terms of energy reduction efficiency, which is rather decent with regard to the ”short” exploration stage (5 minutes).

[17]Running at 2.53 GHz.

[18]To simplify the implementation, the 6 sensor tasks with the same WCET (0.3 ms) and period (15 ms) [Gro10] are combined into one task.

[19]To simplify the implementation, the 2 sensor tasks with the same WCET (0.6 ms) and period (15 ms) [Gro10] are combined into one task.

| Task | $\tau_1$ | $\tau_2$ | $\tau_3$ | $\tau_4$ | $\tau_5$ | $\tau_6$ |
|---|---|---|---|---|---|---|
| Core allocation | $O_1$ | $O_2$ | $O_2$ | $O_2$ | $O_1$ | $O_1$ |
| Speed assignment | $S_2^1$ | $S_2^2$ | $S_1^2$ | $S_2^2$ | $S_2^1$ | $S_2^1$ |

Table 8.12: Computed allocation and speed assignment of the tasks

## 8.4 Chapter Summary

This chapter provided detailed evaluation of the proposed algorithms and approaches. In short, the evaluation is performed in two steps. The first part concentrates on synthetic test cases, where statistically relevant results are obtained. Hereby the evaluation addressed various aspects and mainly answered the following questions.

- Energy reduction efficiency: How much energy can the proposed algorithms save?
- Algorithm run-time: How long will the proposed algorithms take, i.e., how many iterations will the exploration stage need?
- Estimation accuracy: How accurate is the proposed solution quality estimation mechanism?
- ES-AS run-time overhead: How much is the online overhead incurred by the ES-AS approach?
- Impact of CP and SIP: What is the impact of CP and SIP?

In the second part, the evaluation focuses on real-life case studies running on real target platforms. Two experiments demonstrate the applicability of the proposed algorithms and approaches. More concretely, the first experiment addresses single-core processor platforms by focusing on the BeagleBoard with ORCOS, whereas the second experiment is conducted on the Intel Core 2 Duo P9500 processor (a multi-core processor platform) with RTLinux.

# Chapter 9

# Conclusion and Future Work

There is no doubt that energy consumption has become one of the most important design concerns in electronic system development, especially for battery-driven devices. From the system level point of view, there exist two well-established energy reduction techniques: DPM and DVS, which are able to dynamically adjust system energy consumption at run-time. However, this energy reduction comes at the cost of performance loss, which is a crucial point if running tasks have timing constraints. This work exactly addresses this problem and proposes solutions for energy efficient scheduling in hard real-time systems. This chapter concludes the dissertation by summarizing the main contributions and gives an outlook for future work.

## 9.1 Conclusion

In general, the main objective of this dissertation is to optimize system energy consumption by taking the advantage of DPM and DVS while meeting all the task deadlines. Hereby there are several important requirements, which are defined in Chapter 1. In what follows, these requirements are reviewed by showing how they are fulfilled with the main contributions in this work.

- The first requirement is system schedulability, where all the tasks must meet their deadlines. This is solved by the proposed ES-AS approach, which is divided into two stages: exploration stage (ES) and application stage (AS). In ES, the main job is to explore the solution space of a given problem, i.e., one candidate solution in one hyper period. After ES is finished, the best solution found is then applied in AS. This contribution is described in Chapter 6, where its correctness is explicitly proven in terms of system schedulability.
- The second requirement is system-wide energy optimization. In order to address this point, a system model and a problem formulation

are first formally introduced in Chapter 3. Unfortunately, the defined problem is proven to be $\mathcal{NP}$-hard in the strong sense. Thus, Chapter 4 proposes the HSASC and HSAMC algorithms to fulfill this requirement. Both algorithms are extended simulated annealing by incorporating problem domain specific knowledge. In particular, concrete guidelines in terms of penalty and reward values are defined for neighbor selection, which can significantly accelerate the algorithm termination speed. In order to efficiently estimate the algorithm performance and derive a termination criterion, Chapter 5 further provides a regression based mechanism, which is able to estimate solution values at run-time.

- The third requirement is consideration of state switching overhead. There are two types of switching overhead: DPM overhead and DVS overhead. The former is addressed in the proposed ES-AS approach. More specifically, the event recording activity and the idle time prediction solve this requirement for the RTSC and RTMC problems, respectively. In the context of DVS overhead, Chapter 7 explicitly deals with the problem, especially for RTMC. As solutions, a speed inheritance protocol and a conservative protocol are proposed. Both protocols provide sufficient schedulability analysis, which is extended from the traditional schedulability test.

- The next requirement is online. This point is clearly solved by the ES-AS approach, which runs the HSASC and HSAMC algorithms in a completely online fashion. The main challenge hereby is to integrate an iterative search algorithm into the execution of hard real-time tasks. For this, the ES-AS approach maps one iteration of the algorithm to one hyper period of the execution. As a result, the approach is able to consider dynamic changes, such as dynamic slack of tasks, a task joining or leaving a system.

- The final requirement is consideration of multi-core processors. For this, the dissertation addresses a general form of multi-core processors with regard to the DPM and DVS capabilities, namely cluster-based multi-core processors. Its power model is presented in Chapter 3. In short, hereby the processor cores are grouped into clusters, where the cores in the same cluster can only operate at the same speed at the same time. According to this constraint, Chapter 4 proposes special rules for neighbor selection in the HSAMC algorithm and Chapter 6 explicitly describes the application of ES-AS approach for this type of multi-core processors.

Besides, a state-of-the-art analysis is given in Chapter 2, where the main contributions of this dissertation are highlighted in comparison with related work in terms of several important aspects. Of course, the proposed algorithms and approaches are thoroughly evaluated in Chapter 8 to justify their efficiency and applicability.

Before the future work is discussed, a list of relevant publications is summarized as follows: [HM13a], [HM13b], [HM12a], [HM12b] and [HM12c].

## 9.2 Open Issues and Future Work

In general, this work addresses the problem of energy efficient scheduling for hard real-time systems. However, there are several assumptions and limitations in the dissertation. In what follows, they are roughly discussed and some ideas are proposed for future work.

The first limitation is that this work addresses only independent tasks. However, systems with dependent tasks are quite usual as well in practice. In principle, there are two difficulties by considering dependent tasks in the proposed ES-AS approach. The first one is due to schedulability test, which has to be made at each hyper period boundary to ensure that the newly generated solution is feasible. Unlike the test for EDF or RM with independent tasks, hereby the check can not be solved efficiently and thus will incur significant run-time overhead. For instance, there exists one elegant solution by transforming a set of dependent tasks to a set of independent tasks [CSB90], which then can be scheduled by EDF. Hereby task release time and deadline need to be modified accordingly. Because the transformation requires knowledge of task WCET, which changes as speed assignment changes, this modification needs to be executed for each newly generated solution. Clearly, this will significantly increase the run-time overhead at each hyper period boundary. Nevertheless, this process is after all deterministic and will not harm system real-time constraint, provided that the increased overhead is affordable. The second difficult comes from the application on multi-core processors. In fact, partitioning a set of dependent tasks is a fairly tough job and belongs to another research area. Basically, if a solution with an efficient schedulability test exists, then it can also be applied in the ES-AS approach.

Furthermore, the second limitation is that the RTMC problem ignores I/O devices. This is also mainly due to the complexity of task partition, where resource sharing needs to be considered. Similar to the task partition with dependent tasks, this problem is out of the scope of the dissertation. Nevertheless, if there is a solution with an efficient schedulability test, it can also be applied in the ES-AS approach.

In the RTSC problem, it is assumed that there is no critical section by device access, i.e., there is no need for device lock and task blocking. However, if critical sections are present, the problem can be solved by using EDF with priority inheritance protocol. The proposed ES-AS approach and the HSASC algorithm are in fact independent of real-time scheduling strategies, as long as an efficient schedulability test is available, i.e., they are compatible to EDF with priority inheritance protocol.

The HSAMC algorithm is provided in Chapter 4 and the neighbor generation process is illustrated in Figure 4.2. There are three steps to select a neighbor solution: i) select a task, ii) reallocate the task to a new processor core and iii) reassign the task with a new frequency. Instead, there is another possibility to perform this work by changing the order of the last two steps. As already indicated in Chapter 4, in this case the reallocation step is dependent on the result of the reassignment step, if a multi-core processor with heterogeneous cores in terms of power model is considered. As part of the future work, this possibility can be investigated and compared with the version in this dissertation. For this, there is then a need to adapt the guide rules, i.e., redefining the award function. For instance, one may associate each available frequency (note that not each processor core) with a reward value by taking into consideration on how many processor cores it is supported.

Another assumption is related to the power consumption of DVS state switching, which has been ignored so far. Note that Chapter 7 considers only the switching latency, because it is more crucial for hard real-time constraints. In future work, however, the corresponding power consumption can be taken into account by extending the calculation of break-even time (7.14).

There is an issue in Chapter 8 related to the accuracy of the proposed mechanism for performance estimation. More concretely, Figure 8.21 shows a trend that the accuracy becomes worse as the task number increases. This can be improved by adapting the calculation of the smoothing constant $\alpha$ in (5.12) by adding more influence of the task number $n$. For instance, one possible solution is shown as follows.

$$\alpha = \frac{1}{n^2 \cdot o \cdot \exp(\frac{\sum_{i=1}^{n} |F(assign(\tau_i)) - F(SC(\tau_i))|}{n})} \tag{9.1}$$

Clearly, now $\alpha$ becomes more smaller, as $n$ grows. As a result, the termination speed is more slowed down and more time is used for solution search. Surely this will help finding better solutions and subsequently improve the estimation accuracy. However, this adaptation comes at the cost of run-time increase. In brief, a more detailed investigation regarding the computation of $\alpha$ can be carried out in the future.

# Appendices

# Appendix A

# Additional Evaluation Results

## A.1 Energy Reduction Efficiency (Static Slack)

This section shows additional simulation results in terms of energy reduction efficiency for multi-core processor platforms. More specifically, Figure A.1, A.2, A.3 and A.4 show the comparison of different algorithms with regard to the size of task sets. In the figures, the x-axis shows the number of tasks and the y-axis gives the power consumption of output solution normalized to the solution of the LA+LTF+FF algorithm. Clearly, `HSAMC0.01` achieved the best results in all the cases and the smaller the $\beta$, the more power saving can be obtained.

## A.2 Energy Reduction Efficiency (Dynamic Slack)

This section shows further evaluation results in terms of energy reduction efficiency concerning dynamic slack. Figure A.5, A.6, A.7 and A.8 show the comparison using different $\gamma$ value on different platforms. Hereby the x-axis shows the size of task sets and the y-axis gives the normalized power consumption with regard to the `HSAMC0.01` algorithm, where no dynamic slack is present. Obviously, the larger the $\gamma$, the more power can be saved, because more dynamic slack can be utilized. Besides, the power reduction is independent upon task number.

## A.3 Run-Time Analysis

This section shows additional simulation results in terms of algorithm run-time analysis for multi-core processor platforms. More concretely, Figure A.9, A.10, A.11 and A.12 show the comparison of different algorithms with regard to the size of task sets. In the figures, the x-axis shows the number

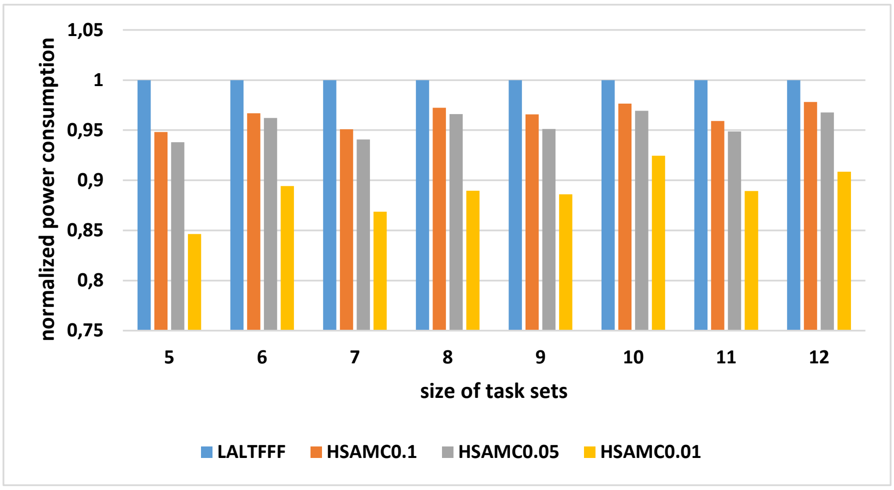


Figure A.1: Energy reduction efficiency comparison among different algorithms in terms of task number (`I2`)

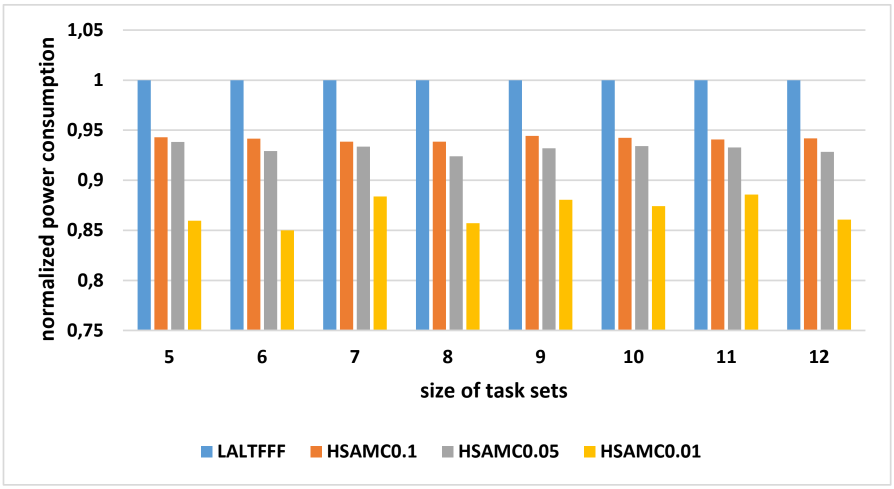


Figure A.2: Energy reduction efficiency comparison among different algorithms in terms of task number (`(I1)`$_2$)

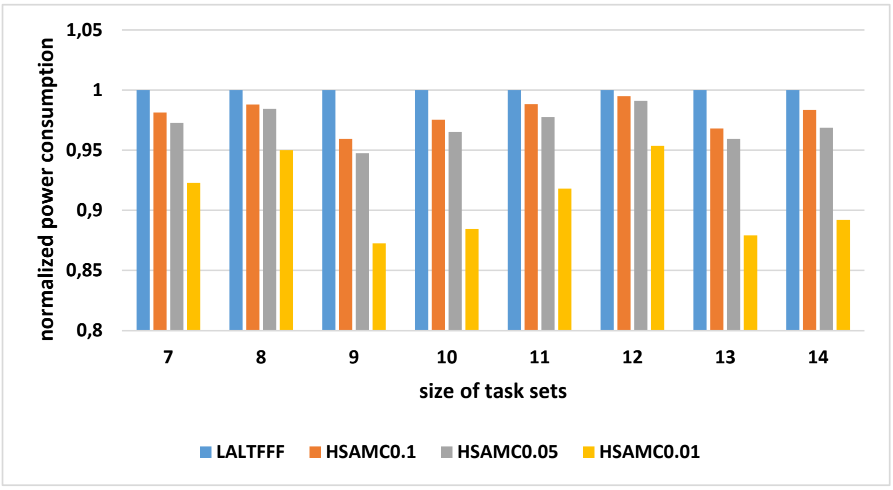


Figure A.3: Energy reduction efficiency comparison among different algorithms in terms of task number (`I4`)

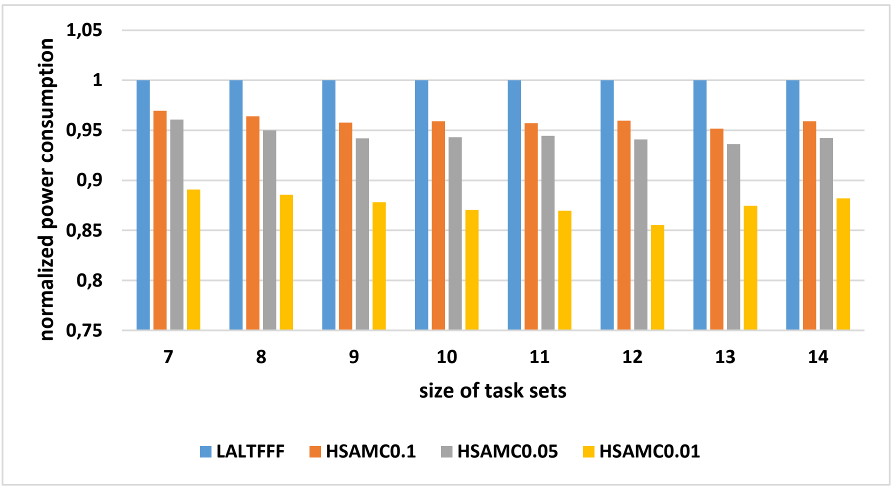


Figure A.4: Energy reduction efficiency comparison among different algorithms in terms of task number (`(I1)4`)

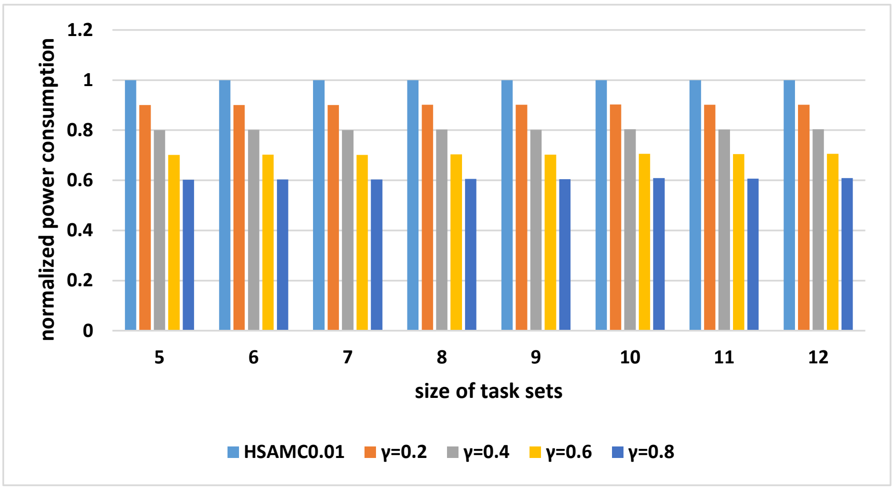


Figure A.5: Average power consumption comparison using different γ in terms of task number (`I2`)

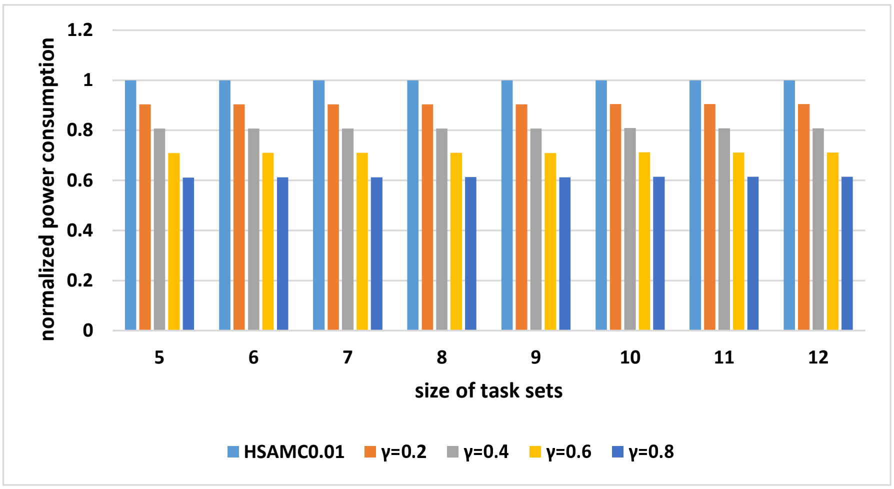


Figure A.6: Average power consumption comparison using different γ in terms of task number (`(I1)`$_2$)

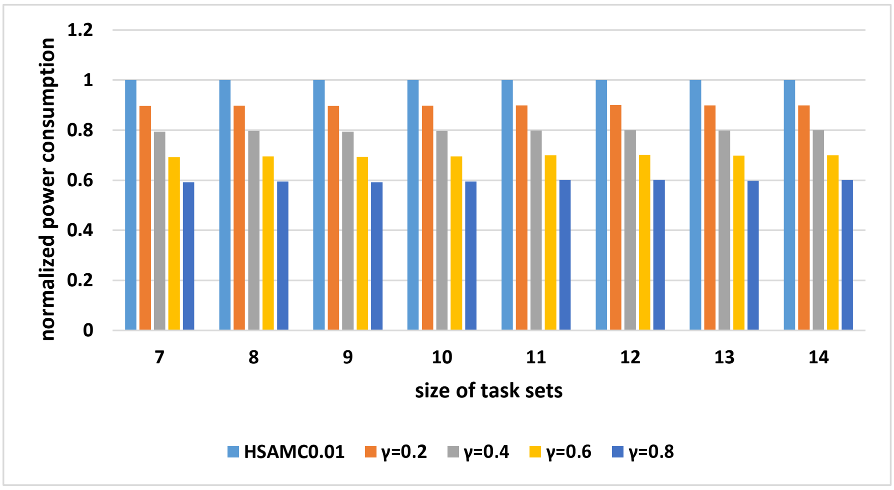


Figure A.7: Average power consumption comparison using different γ in terms of task number (`I4`)

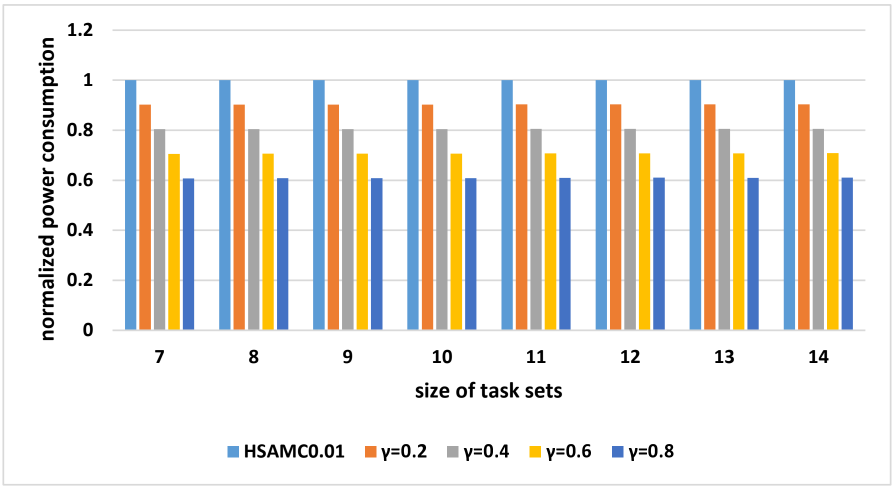


Figure A.8: Average power consumption comparison using different γ in terms of task number (`(I1)4`)

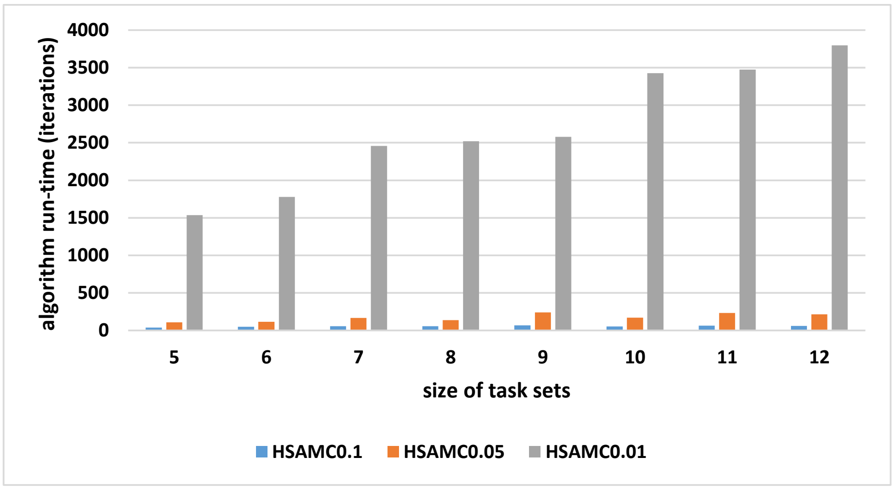


Figure A.9: Run-time comparison among different algorithms in terms of task number (`I2`)

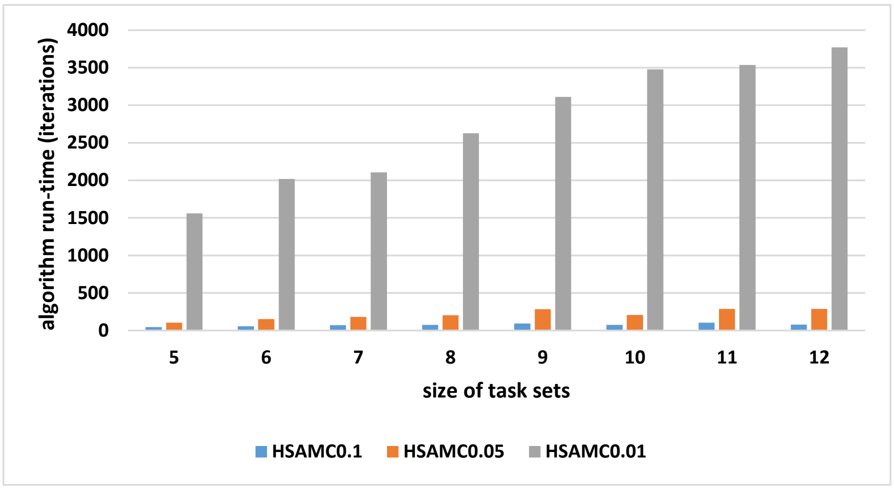


Figure A.10: Run-time comparison among different algorithms in terms of task number (`(I1)2`)

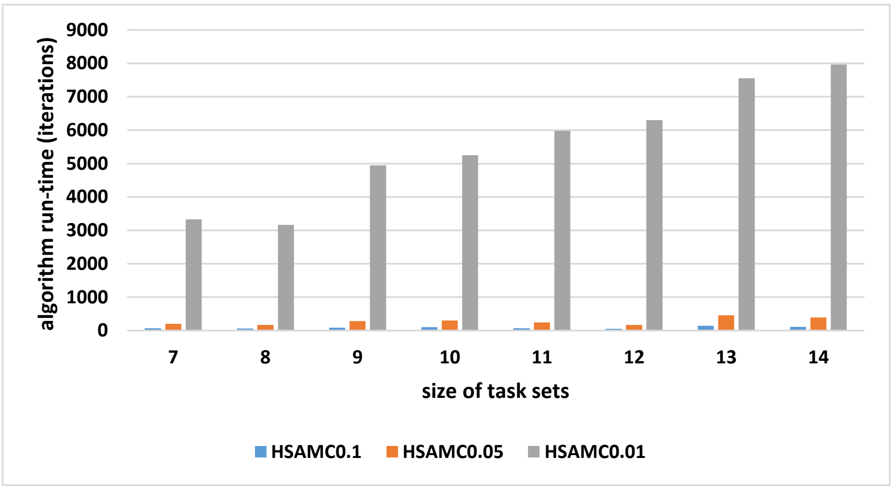


Figure A.11: Run-time comparison among different algorithms in terms of task number (I4)

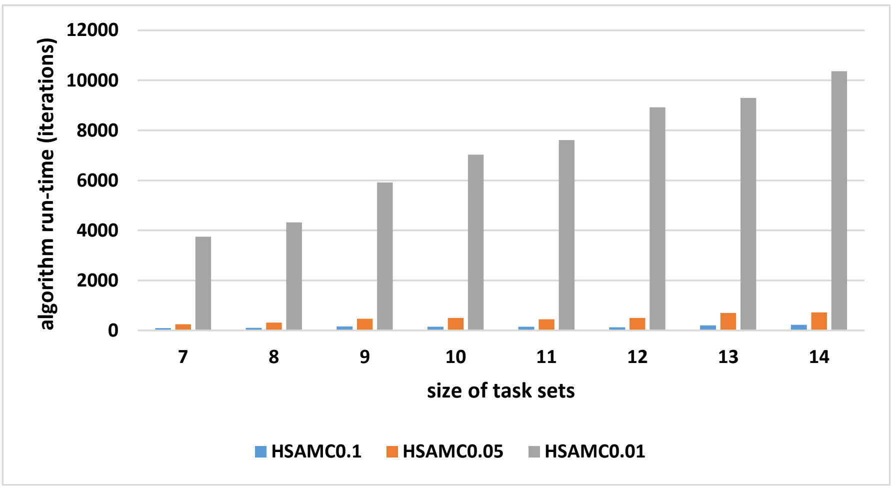


Figure A.12: Run-time comparison among different algorithms in terms of task number ((I1)4)

of tasks and the y-axis gives the number of iterations required by algorithms. Clearly, the smaller the $\beta$, the longer the algorithm takes and the algorithm run-time increases linearly as the size of task sets increases.

# List of Figures

# List of Tables

# List of Algorithms

# List of Abbreviations

| | |
|---|---|
| AA | Algorithm Activity |
| ABS | Anti-lock Braking System |
| ACPI | Advanced Configuration & Power Interface |
| ACU | Airbag Control Unit |
| AEC | Active Energy Consumption |
| AET | Actual Execution Time |
| AS | Application Stage |
| BF | Best Fit |
| CP | Conservative Protocol |
| DA | DPM Activity |
| DAG | Directed Acyclic Graph |
| DPM | Dynamic Power Management |
| DVS or DVFS | Dynamic Voltage and Frequency Scaling |
| EDF | Earliest Deadline First |
| ES | Exploration Stage |
| ESC | Electric Stability Control |
| FF | First Fit |
| FPTAS | Fully Polynomial Time Approximation Scheme |
| HSAMC | Heuristic Search Algorithm for RTMC |
| HSASC | Heuristic Search Algorithm for RTSC |
| ICP | Intel-Core Preemption |
| NF | Next Fit |

| | |
|---|---|
| ORCOS | Organic Reconfigurable Operating System |
| RM | Rate Monotonic |
| RTMC | The Real-Time Energy Optimization Problem on Multi-Core Processor Platforms |
| RTOS | Real-Time Operating System |
| RTSC | The Real-Time Energy Optimization Problem on Single-Core Processor Platforms |
| SA | Simulated Annealing |
| SIP | Speed Inheritance Protocol |
| TGFF | Task Graph For Free |
| VCD | Value Change Dump |
| WCET | Worst Case Execution Time |
| WF | Worst Fit |

# List of Symbols

| | |
|---|---|
| $AFC_i(t)$ | The actually finished cycles of $\tau_i$ until the time point $t$ |
| $CS(\tau_i)$ | The critical speed state of $\tau_i$ on a single-core processor |
| $CS^j(\tau_i)$ | The critical speed state of $\tau_i$, if $\tau_i$ is partitioned onto $O_j$ in a multi-core processor |
| $C_i$ | The $i$-th power state (DPM state) of a single-core processor |
| $C_i^j$ | The $i$-th power state (DPM state) of the $j$-th core in a multi-core processor |
| $D_i^j$ | The $i$-th power state (DPM state) of the $j$-th device |
| $Dev(\tau_i)$ | The set of required devices by $\tau_i$ |
| $EFC_i(t)$ | The expected finished cycles of $\tau_i$ until the time point $t$ |
| $E_{active}(\tau_i, S)$ | The energy consumed by the processor and all the required devices during the execution of $\tau_i$ at a particular speed state $S$ |
| $F(S_i)$ | The operating speed of $S_i$ |
| $F(S_i^j)$ | The operating speed of $S_i^j$ |
| $G_i$ | The $i$-th cluster in a multi-core processor |
| $HP$ | The hyper period of a task set |
| $I_{th}$ | The upper bound to the iteration number specified by the user to stop the algorithm |
| $J : \Omega \rightarrow \mathbb{R}$ | The cost function |
| $L_p$ | The uniform switching latency among the processor P-states (DVS states) |
| $N(\omega)$ | The set of neighbors of the solution $\omega$ |
| $O_i$ | The $i$-th core in a multi-core processor |
| $P(C_i)$ | The power consumption of $C_i$ |
| $P(C_i^j)$ | The power consumption of $C_i^j$ |

| | |
|---|---|
| $P(D_i^j)$ | The power consumption of $D_i^j$ |
| $P(S_i)$ | The power consumption of $S_i$ |
| $P(S_i,S_j)$ | The power consumption of the switching between $S_i$ and $S_j$ |
| $P(S_i^j)$ | The power consumption of $S_i^j$ |
| $P_{off\rightarrow on}(C_i)$ | The power consumption of the switching from $C_i$ to $C_0$ |
| $P_{off\rightarrow on}(C_i^j)$ | The power consumption of the switching from $C_i^j$ to $C_0^j$ |
| $P_{off\rightarrow on}(D_i^j)$ | The power consumption of the switching from $D_i^j$ to $D_0^j$ |
| $P_{on\rightarrow off}(C_i)$ | The power consumption of the switching from $C_0$ to $C_i$ |
| $P_{on\rightarrow off}(C_i^j)$ | The power consumption of the switching from $C_0^j$ to $C_i^j$ |
| $P_{on\rightarrow off}(D_i^j)$ | The power consumption of the switching from $D_0^j$ to $D_i^j$ |
| $Pr_{accept} : \Omega^2 \times \mathbb{N} \rightarrow [0,1]$ | The probability function for the solution acceptance in the simulated annealing algorithm |
| $R_i$ | The $i$-th device |
| $S_i$ | The $i$-th performance state (DVS state) of a single-core processor |
| $S_i^j$ | The $i$-th performance state (DVS state) of the $j$-th core in a multi-core processor |
| $T(S_i,S_j)$ | The switching latency between $S_i$ and $S_j$ |
| $T(\tau_i)$ | The period of $\tau_i$ |
| $T_{be}(C_i)$ | The break-even time of $C_i$ |
| $T_{be}(C_i^j)$ | The break-even time of $C_i^j$ |
| $T_{be}(D_i^j)$ | The break-even time of $D_i^j$ |
| $T_{off\rightarrow on}(C_i)$ | The switching latency from $C_i$ to $C_0$ |
| $T_{off\rightarrow on}(C_i^j)$ | The switching latency from $C_i^j$ to $C_0^j$ |
| $T_{off\rightarrow on}(D_i^j)$ | The switching latency from $D_i^j$ to $D_0^j$ |
| $T_{on\rightarrow off}(C_i)$ | The switching latency from $C_0$ to $C_i$ |
| $T_{on\rightarrow off}(C_i^j)$ | The switching latency from $C_0^j$ to $C_i^j$ |
| $T_{on\rightarrow off}(D_i^j)$ | The switching latency from $D_0^j$ to $D_i^j$ |
| $T_t$ | The cooling temperature at the time or iteration $t$ |
| $U$ | The processor utilization of the single-core processor |

| | |
|---|---|
| $U(O_j)$ | The processor utilization of the $j$-th core in a multi-core processor |
| $U(\tau)$ | The utilization of the task $\tau$ |
| $U_B$ | The processor utilization upper bound on single-core processors to ensure system schedulability |
| $U_B^j$ | The processor utilization upper bound on the $j$-th core in a multi-core processor to ensure the system schedulability on that core |
| $W(\tau_i)$ | The worst case execution time of $\tau_i$ |
| $\Gamma$ | A set of real-time tasks |
| $\Omega$ | The solution space of the simulated annealing algorithm |
| $\Omega^*$ | The set of the optimal solutions |
| $\beta$ | The user parameter specifying the required approximation ratio to stop the algorithm |
| $\varepsilon(\omega)$ | The approximation ratio of the solution $\omega$ |
| $\gamma$ | The factor of task run-time variation |
| $\mathcal{C}$ | The set of power states (DPM states) of a single-core processor |
| $\mathcal{C}^j$ | The set of power states (DPM states) of the $j$-th core in a multi-core processor |
| $\mathcal{D}^j$ | The set of power states (DPM states) of the $j$-th device |
| $\mathcal{G}$ | The set of clusters in a multi-core processor |
| $\mathcal{O}$ | The set of cores in a multi-core processor |
| $\mathcal{R}$ | The set of I/O devices |
| $\mathcal{S}$ | The set of performance states (DVS states) of a single-core processor |
| $\mathcal{S}^j$ | The set of performance states (DVS states) of the $j$-th core in a multi-core processor |
| $\omega$ | A solution in $\Omega$ |
| $\omega_{init}$ | The initial solution of the simulated annealing algorithm |
| $\rho_1$ | The HP-overhead |
| $\rho_2$ | The SC-overhead |
| $\rho_{CP}$ | The false negative error rate of CP |
| $\rho_{SIP}$ | The false negative error rate of SIP |
| $\tau_i$ | A real-time task with index $i$ |
| $alloc : \Gamma \to \mathcal{O}$ | The function partitioning the tasks to the processor cores |

| | |
|---|---|
| $assign : \Gamma \rightarrow S$ | The function assigning the speeds to the tasks |
| $group$ | The function grouping the processor cores to clusters |
| $pen(\tau)$ | The penalty value of the task $\tau$ |
| $prob(O_i)$ | The selection probability of the processor core $O_i$ |
| $prob(\tau)$ | The selection probability of the task $\tau$ |
| $rew(O)$ | The reward value of the processor core $O$ |

# Bibliography

## List of Publications

[HM12a] D. He and W. Müller. “A Heuristic Energy-Aware Approach for Hard Real-Time Systems on Multi-Core Platforms”. In: *Proceedings of the 15th IEEE Euromicro Conference on Digital System Design (DSD)*. **Best Paper Nomination**. 2012.

[HM12b] D. He and W. Müller. “Enhanced Schedulability Analysis of Hard Real-Time Systems on Power Manageable Multi-Core Platforms”. In: *Proceedings of the 9th IEEE Int. Conference on Embedded Software and Systems (ICESS)*. 2012.

[HM12c] Da He and Wolfgang Müller. “Online Energy-Efficient Hard Real-Time Scheduling for Component Oriented Systems”. In: *Proceedings of the 15th IEEE International Symposium on Object-/Component-/Service-Oriented Real-Time Distributed Computing (ISORC)*. **Best Paper Award**. 2012.

[HM13a] D. He and W. Müller. “An Energy-Efficient Heuristic for Hard Real-Time System on Multi-Core Processors”. In: *Proceedings of International Conference on Applied Computing (AC)*. 2013.

[HM13b] Da He and Wolfgang Müller. “A heuristic energy-aware approach for hard real-time systems on multi-core platforms”. In: *Microprocessors and Microsystems* 37.8 (2013), pp. 858–870.

[HMM11] D. He, F. Mischkalla, and W. Müller. “A SysML-based Framework with QEMU-SystemC Code Generation”. In: *Proceedings of 1st international QEMU Users Forum*. 2011.

[MHM10a] F. Mischkalla, D. He, and W. Müller. “A UML Profile for SysML-Based Comodeling for Embedded Systems Simulation and Synthesis”. In: *Proceedings of 1st Workshop on Model Based Engineering for Embedded Systems Design*. 2010.

[MHM10b] F. Mischkalla, D. He, and W. Müller. “Closing the Gap between UML-based Modeling and Simulation of Combined HW/SW Systems”. In: *Proceedings of Design, Automation and Test in Europe (DATE)*. 2010.

[MHM11] F. Mischkalla, D. He, and W. Müller. “A Retargetable SysML-based Front-End for High-Level Synthesis”. In: *Proceedings of 2nd Workshop on Model Based Engineering for Embedded Systems Design*. 2011.

[Mul+10] Wolfgang Müller et al. “The SATURN Approach to SysML-Based HW/SW Co-design”. In: *Proceedings of IEEE Computer Society Annual Symposium on VLSI*. 2010.

[Van+11] Yves Vanderperren et al. “Extending UML for Electronic Systems Design: A Code Generation Perspective”. In: *Design Technology for Heterogeneous Embedded Systems*. Springer, 2011.

## List of References

[AA05] T. A AlEnawy and H. Aydin. "Energy-aware task allocation for rate monotonic scheduling". In: *Proceddings of the 11th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS)*. Mar. 2005, pp. 213–223.

[AB02] James H. Anderson and Sanjoy K. Baruah. "Energy-Aware Implementation of Hard-Real-Time Systems Upon Multiprocessor Platforms". In: *In Proceedings of the 16th International Conference on Parallel and Distributed Computing Systems (ISCA)*. 2002, pp. 430–435.

[Abo+03] Nevine AbouGhazaleh et al. "Collaborative operating system and compiler power management for real-time applications". In: *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'03)*. 2003, pp. 133–141.

[ACP] ACPI. *ACPI Specification*. URL: `http://www.acpi.info/`.

[AG] ELV Elektronik AG. *User Manual of Energy Master Basic*. URL: `http://www.elv-downloads.de/Assets/Produkte/9/994/99434/Downloads/99434_energymaster_basic_um.pdf`.

[Aga+06] Amit Agarwal et al. "Leakage Power Analysis and Reduction for Nanoscale Circuits". In: *IEEE Micro* 26.2 (2006), pp. 68–80.

[AMD] AMD. *AMD Product Specification*. URL: `http://support.amd.com/`.

[ARM] ARM. *ARM big.LITTLE Processing*. URL: `http://www.arm.com/products/processors/technologies/biglittleprocessing.php`.

[Ass] IEEE Stanadrs Association. *1666-2011 - IEEE Standard for Standard SystemC Language Reference Manual*. URL: `http://standards.ieee.org/findstds/standard/1666-2011.html`.

[AY03] Hakan Aydin and Qi Yang. "Energy-Aware Partitioning for Multiprocessor Real-Time Systems". In: *Proceedings of the 17th International Symposium on Parallel and Distributed Processing (IPDPS)*. IEEE Computer Society, 2003, p. 113.2.

[Ayd+04] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. "Power-aware scheduling for periodic real-time tasks". In: *IEEE Transactions on Computers* 53.5 (May 2004), pp. 584–600.

[Bac96] T. Bäck. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, 1996.

[BBM00] Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. "A survey of design techniques for system-level dynamic power management". In: *IEEE Transactions on Very Large Scale Integration Systems* 8.3 (2000), pp. 299–316.

[Beaa] BeagleBoard.org. *BeagleBoard*. URL: `http://beagleboard.org/Products/BeagleBoard`.

[Beab] BeagleBoard.org. *BeagleBoard (Rev C4) System Reference Manual*. URL: `http://circuitco.com/support/files/BeagleBoard-RevC3/BeagleBoard_revC3_SRM.pdf`.

[Bey] BeyondTTL. *Value Change Dump (VCD)*. URL: `http://www.beyondttl.com/vcd.php`.

[Bry95] Wlodzimierz Bryc. *The Normal Distribution: Characterizations with Applications*. Springer, 1995.

[BT93] Dimitris Bertsimas and John Tsitsiklis. “Simulated Annealing”. In: *Statistical Science* 8.1 (1993), pp. 10–15.

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

[BW08] D. M. Bates and D. G. Watts. “Nonlinear Regression Analysis and Its Applications”. In: John Wiley & Sons, Inc., 2008. Chap. Nonlinear Regression: Iterative Estimation and Linear Approximations.

[Cer85] V. Černý. “Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm”. In: *Journal of Optimization Theory and Applications* 45 (1 1985), pp. 41–51.

[CG06] H. Cheng and S. Goddard. “Online energy-aware I/O device scheduling for hard real-time systems”. In: *Proceedings of Design, Automation and Test in Europe (DATE)*. 2006.

[CG09] H. Cheng and S. Goddard. “SYS-EDF: a system-wide energy-efficient scheduling algorithm for hard real-time systems”. In: *International Journal of Embedded Systems* (2009).

[Cha+12] Hung-Lin Chao et al. “Congestion-aware scheduling for NoC-based reconfigurable systems”. In: *Proceedings of Design, Automation Test in Europe Conference Exhibition (DATE)*. 2012, pp. 1561–1566.

[Che+04] Jian-Jia Chen et al. “Multiprocessor energy-efficient scheduling with task migration considerations”. In: *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS)*. July 2004, pp. 101–108.

[Che+07] Jian-Jia Chen, Chuan-Yue Yang, Tei-Wei Kuo, and C.-S. Shih. “Energy-Efficient Real-Time Task Scheduling in Multiprocessor DVS Systems”. In: *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE Computer Society, 2007, pp. 342–349.

[CHK06] J.-J. Chen, H.-R. Hsu, and T.-W. Kuo. “Leakage-Aware Energy-Efficient Scheduling of Real-Time Tasks in Multiprocessor Systems”. In: *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE Computer Society, 2006, pp. 408–417.

[CK05] Jian-Jia Chen and Tei-Wei Kuo. “Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics”. In: *Proceedings of the 34th International Conference on Parallel Processing (ICPP)*. 2005, pp. 13–20.

[CK07] Jian-Jia Chen and Chin-Fu Kuo. “Energy-Efficient Scheduling for Real-Time Systems on Dynamic Voltage Scaling (DVS) Platforms”. In: *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 2007, pp. 28–38.

[CR11] K. Chakraborty and S. Roy. “Topologically homogeneous power-performance heterogeneous multicore systems”. In: *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Mar. 2011, pp. 1–6.

[CSB90] H. Chetto, M. Silly, and T. Bouchentouf. “Dynamic scheduling of real-time tasks under precedence constraints”. In: *Real-Time Systems* 2.3 (1990), pp. 181–194. ISSN: 0922-6443.

[CYW10] Yi-Jung Chen, Chia-Lin Yang, and Po-Han Wang. “PM-COSYN: PE and memory co-synthesis for MPSoCs”. In: *Proceedings of Design, Automation Test in Europe Conference Exhibition (DATE)*. 2010, pp. 1590–1595.

[DA08a] V. Devadas and H. Aydin. “Real-Time Dynamic Power Management through Device Forbidden Regions”. In: *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2008, pp. 34–44.

[DA08b] Vinay Devadas and Hakan Aydin. “On the interplay of dynamic voltage scaling and dynamic power management in real-time embedded applications”. In: *Proceedings of the 8th ACM international conference on Embedded software*. ACM, 2008, pp. 99–108.

[DA10] V. Devadas and H. Aydin. “DFR-EDF: A Unified Energy Management Framework for Real-Time Systems”. In: *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2010.

[DB11] Robert I. Davis and Alan Burns. “A survey of hard real-time scheduling for multiprocessor systems”. In: *ACM Computing Surveys* 43.4 (2011), 35:1–35:44.

[Des92] M. P. Desai. “An eigenvalue-based approach to the finite time behavior of simulated annealing”. PhD thesis. University of Illinois at Urbana-Champaign, 1992.

[Des99] M. P. Desai. “Some results characterizing the finite time behaviour of the simulated annealing algorithm”. In: *Sadhana* 24 (4-5 1999), pp. 317–337.

[Dev03] U.C. Devi. “An improved schedulability test for uniprocessor periodic task systems”. In: *Proceedings of the 15th Euromicro Conference on Real-Time Systems*. 2003, pp. 23–30.

[DNM06] D. Dal, A. Nunez, and N. Mansouri. “Power islands: a high-level technique for counteracting leakage in deep sub-micron”. In: *Proceedings of the 7th International Symposium on Quality Electronic Design (ISQED)*. Mar. 2006.

[Dor92] M. Dorigo. “Optimization, Learning and Natural Algorithms”. PhD thesis. Politecnico di Milano, Italy, 1992.

[DS86] Fabio Romeo Debasis Mitra and Alberto Sangiovanni-Vincentelli. “Convergence and Finite-Time Behavior of Simulated Annealing”. In: *Advances in Applied Probability* 18.3 (1986), pp. 747–771.

[Edm65] Jack Edmonds. “Paths, trees, and flowers”. In: *Canadian Journal of Mathematics* 17 (1965), pp. 449–467.

[ES03] Agoston E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. 1st. Springer, 2003.

[FN08] Ge Fen and Wu Ning. “A minimum-path mapping algorithm for 2D mesh Network on Chip architecture”. In: *Proceedings of the IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*. 2008, pp. 1542–1545.

[FZ08] G. Fettweis and E. Zimmermann. “ICT Energy Consumption - Trends and Challenges”. In: *The 11th International Symposium on Wireless Personal Multimedia Communications (WPMC)*. 2008.

[Gan+12] Junhe Gan, P. Pop, F. Gruian, and J. Madsen. “Robust and flexible mapping for real-time distributed applications during the early design phases”. In: *Proceedings of Design, Automation Test in Europe Conference Exhibition (DATE)*. 2012, pp. 935–940.

[GC04] Michael A. Golberg and Hokwon A. Cho. *Introduction to Regression Analysis*. WIT Press, 2004.

[GJ75] M. R. Garey and D. S. Johnson. “Complexity Results for Multiprocessor Scheduling under Resource Constraints”. In: *SIAM J. on Computing* 4.4 (1975), pp. 397–411.

[GJ78] M. R. Garey and D. S. Johnson. “”Strong” NP-Completeness Results: Motivation, Examples, and Implications”. In: *Journal of the ACM (JACM)* 25.3 (1978), pp. 499–508.

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.

[GK01] F. Gruian and K. Kuchcinski. “LEneS: task scheduling for low-energy systems using variable supply voltage processors”. In: *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2001, pp. 449–455.

[Gro10] Stefan Grösbrink. “Comparison of alternative hierarchical scheduling techniques for the virtualization of embedded real-time systems”. MA thesis. University of Paderborn, 2010.

[Gru00] Flavius Gruian. “System-Level Design Methods for Low-Energy Architectures Containing Variable Voltage Processors”. In: *In Proceedings of the Workshop on Power-Aware Computing Systems (PACS)*. 2000, pp. 1–12.

[GTK] GTKWave. *Welcome to GTKWave*. URL: `http://gtkwave.sourceforge.net/`.

[Han+12] Jian-Jun Han et al. “Synchronization-Aware Energy Management for VFI-Based Multicore Real-Time Systems”. In: *IEEE Transactions on Computers* 61.12 (Dec. 2012), pp. 1682–1696.

[HJJ03] Darrall Henderson, SheldonH. Jacobson, and AlanW. Johnson. “The Theory and Practice of Simulated Annealing”. In: *Handbook of Metaheuristics*. Ed. by Fred Glover and GaryA. Kochenberger. Vol. 57. International Series in Operations Research and Management Science. Springer US, 2003, pp. 287–319.

[Hu+04] Jingcao Hu, Youngsoo Shin, Nagu Dhanwada, and Radu Marculescu. “Architecting voltage islands in core-based system-on-a-chip designs”. In: *Proceedings of the 2004 international symposium on Low power electronics and design (ISLPED)*. ACM, 2004, pp. 180–185.

[Ini] Accellera Systems Initiative. *Accellera Systems Initiative*. URL: http://www.accellera.org/home/.

[Insa] Texas Instruments. *OMAP3530 Power Estimation Spreadsheet*. URL: http://processors.wiki.ti.com/index.php/OMAP3530_Power_Estimation_Spreadsheet.

[Insb] Texas Instruments. *OMAP3530 Technical DOcuments*. URL: http://www.ti.com/product/omap3530.

[Insc] Texas Instruments. *OMAP3530/25 Applications Processor (Rev. F)*. URL: http://www.ti.com/lit/gpn/omap3530.

[Insd] Texas Instruments. *OMAP35x Technical Reference Manual (Rev. X)*. URL: http://www.ti.com/litv/pdf/spruf98x.

[Inse] Texas Instruments. *Texas Instruments*. URL: http://www.ti.com/.

[Inta] Intel. *Intel Core 2 Duo P9500 Processor Datasheet*. URL: http://download.intel.com/design/mobile/datashts/32012001.pdf.

[Intb] Intel. *Intel Core 2 Duo Processor P9500*. URL: http://ark.intel.com/de/products/35566/Intel-Core2-Duo-Processor-P9500-6M-Cache-2_53-GHz-1066-MHz-FSB.

[Intc] Intel. *Intel Processor Specification*. URL: http://www.intel.com/products/processor/core2quad.

[Intd] Intel. *Intel Processor Specification*. URL: http://www.intel.com/products/processor/core2duo/index.htm.

[Inte] Intel. *Intel PXA270 Processor Electrical, Mechanical, and Thermal Specification*.

[ITP] ITProPortal. *Exclusive : ARM Cortex-A15 ”40 Per Cent” Faster Than Cortex-A9*. URL: http://www.itproportal.com/2011/03/14/exclusive-arm-cortex-a15-40-cent-faster-cortex-a9.

[Jac+05] S. H. Jacobson, S. N. Hall, L. A. McLay, and J. E. Orosz. “Performance Analysis of Cyclical Simulated Annealing Algorithms”. In: *Methodology and Computing in Applied Probability* 7 (2 2005), pp. 183–201.

[Jef] Brian Jeff. *Advances in big.LITTLE Technology for Power and Energy Savings*. URL: http://www.arm.com/files/pdf/Advances_in_big.LITTLE_Technology_for_Power_and_Energy_Savings.pdf.

[JG04] R. Jejurikar and R. Gupta. “Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems”. In: *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*. 2004, pp. 78–81.

[JG05] R. Jejurikar and R. Gupta. “Energy aware non-preemptive scheduling for hard real-time systems”. In: *Proceedings of the 17th Euromicro Conference on Real-Time Systems (ECRTS)*. July 2005.

[JHM06] SheldonH. Jacobson, ShaneN. Hall, and LauraA. McLay. “Visiting near-optimal solutions using local search algorithms”. In: *Compstat 2006 - Proceedings in Computational Statistics*. Ed. by Alfredo Rizzi and Maurizio Vichi. Physica-Verlag HD, 2006, pp. 471–481.

[Joh+91] David S. Johnson, Cecilia R. Aragon, Lyle A. McGeoch, and Catherine Schevon. “Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning”. In: *Operations Research* 39.3 (1991), pp. 378–406.

[Kim+02] Woonseok Kim et al. “Performance comparison of dynamic voltage scaling algorithms for hard real-time systems”. In: *Proceedings of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2002, pp. 219–228.

[Kir+83] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. “Optimization by Simulated Annealing”. In: *Science* 220 (1983), pp. 671–680.

[Kon+10] F. Kong, Y. Wang, Q. Deng, and W. Yi. “Minimizing Multi-resource Energy for Real-Time Systems with Discrete Operation Modes”. In: *Proceedings of the 22nd Euromicro Conference on Real-Time Systems (ECRTS)*. 2010, pp. 113–122.

[Kop11] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. 2nd. Springer, 2011.

[KYD11] Fanxin Kong, Wang Yi, and Qingxu Deng. “Energy-Efficient Scheduling of Real-Time Tasks on Cluster-Based Multicores”. In: *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2011.

[KZS11] Tejaswini Kolpe, Antonia Zhai, and Sachin S Sapatnekar. “Enabling improved power management in multicore processors through clustered DVFS”. In: *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Mar. 2011, pp. 1–6.

[Lam+12] Sofie Lambert et al. “Worldwide electricity consumption of communication networks”. In: *Optics Express* 20.26 (Dec. 2012), B513–B524.

[Lee09] W. Y. Lee. “Energy-Saving DVFS Scheduling of Multiple Periodic Real-Time Tasks on Multi-core Processors”. In: *Proceedings of the 13th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. 2009, pp. 216–223.

[Len] Lenovo. *ThinkPad T400 Datasheet*. URL: `http://www.lenovo.com/pdf/us/en/t400_and_t500_datasheet.pdf`.

[LHC11] K. Lampka, K. Huang, and J.-J. Chen. “Dynamic counters and the efficient and effective online power management of embedded real-time systems”. In: *Proceedings of the 7th IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES+ISSS)*. ACM, 2011, pp. 267–276.

[LL73] C. L. Liu and James W. Layland. “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment”. In: *Journal of the ACM* 20.1 (1973), pp. 46–61.

[Loc01] M. Locatelli. “Convergence and first hitting time of simulated annealing algorithms for continuous global optimization”. In: *Mathematical Methods of Operations Research* 54 (2 2001), pp. 171–199.

[LS04] Cheol-Hoon Lee and K.G. Shin. “On-line dynamic voltage scaling for hard real-time systems using the EDF algorithm”. In: *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS)*. 2004, pp. 319–335.

[Ltd] ARM Ltd. *ARM Cortex-A Series*. URL: `http://www.arm.com/products/processors/cortex-a/index.php`.

[Man] Linux Manual. *stress - Linux man page*. URL: `http://linux.die.net/man/1/stress`.

[Mar03] P. Marwedel. *Embedded System Design*. Kluwer, 2003.

[Mil12] M.J. Miller. *Intel Enters Smartphone Chip Race For Real*. 2012. URL: `http://forwardthinking.pcmag.com/ces/292745-intel-enters-smartphone-chip-race-for-real`.

[Moo98] G.E. Moore. “Cramming More Components Onto Integrated Circuits”. In: *Proceedings of the IEEE* 86.1 (Jan. 1998), pp. 82–85.

[MPV12] Douglas C. Montgomery, Elizabeth A. Peck, and G. Geoffrey Vining. *Introduction to Linear Regression Analysis*. John Wiley & Sons, Inc., 2012.

[MRS85] D. Mitra, F. Romeo, and A. Sangiovanni-Vincentelli. “Convergence and finite-time behavior of simulated annealing”. In: *Proceedings of the 24th IEEE Conference on Decision and Control*. Vol. 24. 1985, pp. 761–767.

[MT13] Jérôme Monnot and Sophie Toulouse. “The Traveling Salesman Problem and its Variations”. In: *Paradigms of Combinatorial Optimization*. John Wiley & Sons, Inc., 2013, pp. 173–214.

[Nik+11] AlexanderG. Nikolaev, SheldonH. Jacobson, ShaneN. Hall, and Darrall Henderson. “A framework for analyzing sub-optimal performance of local search algorithms”. In: *Computational Optimization and Applications* 49 (3 2011), pp. 407–433.

[Niu10] L. Niu. “Energy efficient scheduling for hard real-time systems with fixed-priority assignment”. In: *Proceedings of the 29th IEEE International Performance Computing and Communications Conference (IPCCC)*. Dec. 2010, pp. 153–160.

[Niu11] Linwei Niu. “System-level energy-efficient scheduling for hard real-time embedded systems”. In: *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Mar. 2011.

[NJ10] A. G. Nikolaev and S. H. Jacobson. “Simulated Annealing”. In: *Handbook of Metaheuristics*. Ed. by M. Gendreau and J.-Y. Potvin. International Series in Operations Research and Management Science. Springer US, 2010, pp. 1–40.

[NL11] Linwei Niu and Wei Li. “Energy-efficient fixed-priority scheduling for real-time systems based on threshold work-demand analysis”. In: *Proceedings of the 7th international conference on hardware/software codesign and system synthesis (CODES+ISSS)*. 2011.

[NS00] Andreas Nolte and Rainer Schrader. “A Note on the Finite Time Behavior of Simulated Annealing”. In: *Mathematics of Operations Research* 25.3 (2000), pp. 476–484. ISSN: 0364-765X.

[OJ02] Jeffrey E. Orosz and Sheldon H. Jacobson. “Finite-Time Performance Analysis of Static Simulated Annealing Algorithms”. In: *Computational Optimization and Applications* 21.1 (2002), pp. 21–53.

[OMA] OMAPpedia. *Power Management Device Latencies Measurement*. URL: `http://www.omappedia.org/wiki/Power_Management_Device_Latencies_Measurement`.

[Ope] OpenOCD. *Open On-Chip Debugger*. URL: `http://openocd.sourceforge.net/`.

[ORC] ORCOS. *Organic Reconfigurable Operating System*. URL: `https://orcos.cs.uni-paderborn.de`.

[Qua+04] Gang Quan, Linwei Niu, X.S. Hu, and B. Mochocki. “Fixed priority scheduling for reducing overall energy on variable voltage processors”. In: *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS)*. 2004, pp. 309–318.

[QZ08] Xuan Qi and Dakai Zhu. “Power Management for Real-Time Embedded Systems on Block-Partitioned Multicore Platforms”. In: *Proceedings of the 5th International Conference on Embedded Software and Systems (ICESS)*. July 2008, pp. 110–117.

[RDV] David Rhodes, Robert Dick, and Keith Vallerio. *Task Graph For Free*. URL: `http://ziyang.eecs.umich.edu/~dickrp/tgff/`.

[Ric+06] U. Richter et al. “Towards a generic observer/controller architecture for Organic Computing”. In: *GI Jahrestagung*. Ed. by Christian Hochberger and Rüdiger Liskowsky. Vol. 93. LNI. GI, 2006, pp. 112–119.

[Row+08] Anthony Rowe, Karthik Lakshmanan, Haifeng Zhu, and Ragunathan Rajkumar. “Rate-Harmonized Scheduling for Saving Energy”. In: *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. IEEE Computer Society, 2008, pp. 113–122.

[RWep] S. Raman and B. Wah. “Quality-time tradeoffs in simulated annealing for VLSI placement”. In: *Proceedings of the 15th Annual International Computer Software and Applications Conference (COMPSAC)*. Sep, pp. 430–435.

[SC05] V. Swaminathan and K. Chakrabarty. “Pruning-based, energy-optimal, deterministic I/O device scheduling for hard real-time systems”. In: *ACM Transactions on Embedded Computing Systems* 4 (1 Feb. 2005), pp. 141–167.

[Sch05] Hartmut Schmeck. “Organic computing - a new vision for distributed embedded systems”. In: *Proceedings of the 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*. May 2005, pp. 201–203.

[Seo+08] Euiseong Seo, Jinkyu Jeong, Seonyeong Park, and Joonwon Lee. “Energy Efficient Scheduling of Real-Time Tasks on Multicore Processors”. In: *IEEE Transactions on Parallel and Distributed Systems* 19.11 (Nov. 2008), pp. 1540–1552.

[SH88] Galen H. Sasaki and Bruce Hajek. “The time complexity of maximum matching by simulated annealing”. In: *Journal of the ACM* 35.2 (1988), pp. 387–403.

[Shi] Shimpi. *The ARM vs x86 Wars Have Begun: In-Depth Power Analysis of Atom, Krait & Cortex A15*. URL: http://www.anandtech.com/show/6536/arm-vs-x86-the-real-showdown.

[THK] T. Ts’o, D. Hart, and J. Kacur. *Real-Time Linux Wiki*. URL: https://rt.wiki.kernel.org/.

[Tooa] Tin Can Tools. *Flyswatter*. URL: http://www.tincantools.com/wiki/Flyswatter.

[Toob] Tin Can Tools. *Tin Can Tools Homepage*. URL: http://www.tincantools.com/home.php.

[Vaz02] Vijay V. Vazirani. *Approximation Algorithms*. 2nd. Springer, 2002.

[VK83] M.P. Vecchi and S. Kirkpatrick. “Global Wiring by Simulated Annealing”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 2.4 (Oct. 1983), pp. 215–222.

[Wan+12] Tianyi Wang, Gang Quan, Shangping Ren, and Meikang Qiu. “Topology Virtualization for Throughput Maximization on Many-Core Platforms”. In: *Proceedings of IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS)*. 2012, pp. 408–415.

[Xu+04] Ruibin Xu, Chenhai Xi, Rami Melhem, and Daniel Moss. “Practical PACE for embedded systems”. In: *Proceedings of the 4th ACM international conference on Embedded software (EMSOFT)*. 2004, pp. 54–63.

[YCK05] Chuan-Yue Yang, Jian-Jia Chen, and Tei-Wei Kuo. “An approximation algorithm for energy-efficient scheduling on a chip multiprocessor”. In: *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2005, pp. 468–473.

[YDS95] F. Yao, A. Demers, and S. Shenker. “A scheduling model for reduced CPU energy”. In: *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*. Oct. 1995, pp. 374–382.

[YP02] Yang Yu and V. K Prasanna. “Power-aware resource allocation for independent tasks in heterogeneous real-time systems”. In: *Proceedings of the 9th International Conference on Parallel and Distributed Systems (ICPADS)*. Dec. 2002, pp. 341–348.

[ZC02] Fan Zhang and Samuel T Chanson. “Processor Voltage Scheduling for Real-Time Tasks with Non-Preemptible Sections”. In: *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS)*. IEEE Computer Society, 2002.

[ZCK07] Sushu Zhang, Karam S. Chatha, and Goran Konjevod. “Approximation algorithms for power minimization of earliest deadline first and rate monotonic schedules”. In: *Proceedings of the international symposium on Low power electronics and design (ISLPED)*. 2007, pp. 225–230.

[ZMC03] D. Zhu, R. Melhem, and B. R. Childers. “Scheduling with Dynamic Voltage Speed Adjustment Using Slack Reclamation in Multiprocessor Real-Time Systems”. In: *IEEE Transactions on Parallel and Distributed Systems* 14.7 (2003), pp. 686–700.

[ZMG09] H. Zabel, W. Müller, and A. Gerstlauer. “Accurate RTOS Modeling and Analysis with SystemC”. In: *Hardware-dependent Software*. Springer, 2009.

[ZX06] Xiliang Zhong and Cheng-Zhong Xu. “System-Wide Energy Minimization for Real-Time Tasks: Lower Bound and Approximation”. In: *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD’06)*. 2006, pp. 516–521.