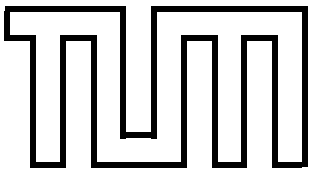


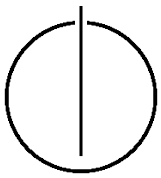
DEPARTMENT OF INFORMATICS

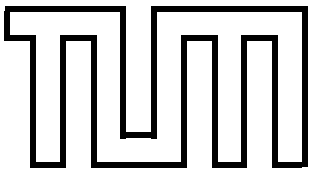
TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

# Genetic Algorithm for Hard Real-time Thermal-aware Scheduling on Hardware Platform

Manuel Dell'Antonio





DEPARTMENT OF INFORMATICS

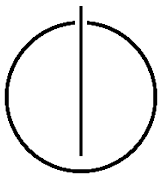
TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

# Genetic Algorithm for Hard Real-time Thermal-aware Scheduling on Hardware Platform

# Genetic Algorithm für wärmebewusstes Echtzeitscheduling auf einer Hardware Plattform

| | |
|---|---|
| Author: | Manuel Dell'Antonio |
| Supervisor: | Prof. Dr.-Ing. habil. Alois Knoll |
| Advisor: | Mingchuan Zhou, M.Sc. |
| Submission Date: | August 15, 2017 |



I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, August 15, 2017 Manuel Dell'Antonio

# Acknowledgments

First and foremost, I would like to thank Mr. Mingchuan Zhou for his advice and assistance he provided throughout the creation of this thesis, as well as Prof. Knoll for the chance to write it at the chair of Robotics and Embedded Systems. Additionally I wish to thank my family and friends for their constant support during my bachelor studies.

# Abstract

The growing demand for sophisticated hardware and software systems has led to the development of high-powered processors. The large number of on-chip components results in an increasingly high power density and consequently a rising power dissipation. These high operating temperatures caused by such development have an adverse effect on reliability and longevity of the system. Especially in the area of real-time systems these factors present a major challenge which continues to interfere with progress.

To tackle these challenges, a variety of different approaches have been studied and implemented. This thesis will examine a novel approach which aims at minimizing the CPU temperature for hard real-time systems with schedules derived by a genetic algorithm. The schedules attempt to achieve this by scaling the processor frequency and voltage throughout their execution. With the support of a scheduling framework the results given by this approach will be tested alongside other state-of-the-art thermal management methods. The experiments will be performed on two different hardware platforms, in an environment designed to simulate an actual hard real-time system. Based on temperature measurements taken during the experiments the performance of the genetic algorithm will be evaluated and the effectiveness of its schedules compared to other approaches. Ultimately an assessment of the benefits and shortcomings of this new approach will be provided.

# Contents

**Acknowledgments** **v**

**Abstract** **vi**

**1. Introduction** **1**
1.1. Motivation . . . . . 1
1.2. Objectives . . . . . 2
1.3. Outline . . . . . 2

**2. Background** **3**
2.1. Hard Real-Time Scheduling . . . . . 3
2.2. Dynamic Thermal Management . . . . . 4
2.2.1. Dynamic Power Management . . . . . 4
2.2.2. Dynamic Voltage and Frequency Scaling . . . . . 6
2.3. Genetic Algorithm . . . . . 9
2.3.1. GMPT . . . . . 9
2.3.2. Boxplots . . . . . 11

**3. Approach** **13**
3.1. Framework . . . . . 13
3.1.1. Architecture . . . . . 13
3.1.2. Platform Adjustments . . . . . 15
3.1.3. Overhead Analysis . . . . . 17
3.2. Experiments . . . . . 19
3.2.1. Platform Frequency Ranges . . . . . 20
3.2.2. Thermal Profiles . . . . . 21

**4. Results** **25**
4.1. Platform: Notebook . . . . . 25
4.1.1. Single-Task . . . . . 25
4.1.2. Multi-Task . . . . . 27
4.2. Platform: Raspberry Pi 3B . . . . . 29
4.2.1. Single-Task . . . . . 29

4.2.2. Multi-Task . . . . . 31
4.3. Assessment . . . . . 33

**5. Conclusion** **35**

**6. Future Work** **37**

**A. Schedules** **39**

**List of Figures** **43**

**List of Tables** **44**

**Bibliography** **45**

# 1. Introduction

## 1.1. Motivation

The fast paced progress in the area of hardware and software systems that has been witnessed over the last half-century has allowed them to transcend their original limited application fields. Available processing power has steadily increased for the last decades and with it, new challenges have surfaced. Even though consistently more components have been packed into integrated circuits, their size has mostly remained the same or, in some cases, even shrunk. This has facilitated the integration of high powered processors in many applications but as a result large amounts of power dissipation can be observed, mostly in the form of heat. This represents a major challenge in embedded systems which are often very limited in their use of hardware cooling devices due to their environment. Recent developments in fields like the automotive industry have sparked a high demand for powerful CPUs to be used for applications such as autonomous driving and real-time obstacle recognition, forcing these circuits to their thermal limits. A particularly tough challenge is faced by safety critical applications since they are required to guarantee a certain quality of service under all circumstances, thus a careful balance between performance and temperature has to be found. The common temperature control approaches used in many of these systems today likely won't be able to satisfy the thermal and performance requirements of the future. In order to keep up with the progress of available processing power current approaches to temperature control have to be reexamined and new ideas evaluated.

Numerous approaches to tackle this problem on a software level have been studied (see chapter 2) and one novel idea will be analyzed in the course of this thesis. Zhou developed GMPT [34], a method that employs a genetic algorithm to find a static schedule which minimizes the processor peak temperature by altering the CPU frequency throughout the schedule while ensuring that all jobs still meet their deadline. To get a better understanding of the advantages and drawbacks of GMPT, a study on it has been conducted, including experiments on two hardware platforms under conditions which closely resemble real-time systems.

## 1.2. Objectives

The overall goal of this thesis is the analysis and evaluation of the GMPT algorithm. The individual objectives are summarized as follows:

- Extraction and assessment of the thermal behavior of a notebook platform. This includes adaption of the employed framework and environment to minimize potential interference which may falsify the results.
- Extraction and assessment of the thermal behavior of a Raspberry Pi 3B. The same framework that has been used on the notebook will be employed here, however, some code rework is required to maintain equal functionality on this new platform.
- Execution of frequency scaling schedules which were computed by different algorithms. The results obtained by running these experiments will serve to evaluate the GMPT algorithm and compare it to a similar approach. An assessment of the schedules generated by GMPT will ultimately be provided.

## 1.3. Outline

The remainder of this thesis is organized in five chapters. Chapter 2 provides background information on various concepts which form the basis of this thesis. It introduces ideas which constitute the starting points for the presented work and gives an overview of related work. Chapter 3 gives insight into the environment used for the experiments, the adjustments that had to be made to it and illustrates the groundwork that was required to successfully run the experiments. Chapter 4 presents the data that was gathered from the experiments, compares the results for the different algorithms and finally assesses the performance of both approaches. Chapter 5 provides a summary of the contributions of this thesis. Finally, Chapter 6 concludes this thesis by giving an outlook into the possible future developments of GMPT and Dynamic Thermal Management as a whole.

# 2. Background

This chapter provides background information on topics which are essential for the understanding of the work presented in this thesis. First, an overview of hard real-time systems and their classification is presented, additionally examining the basics of scheduling techniques used in these systems. Then, an introduction into Dynamic Power Management methods is given, explaining their overall functionality and providing references to related research into the respective subdomains. The concept of genetic algorithms is introduced and the GMPT algorithm, upon which the experiments of this thesis are based on, is illustrated. Finally, boxplots are explained and advice on how to interpret the data displayed by them is provided since they are used throughout this thesis to illustrate experimental results.

## 2.1. Hard Real-Time Scheduling

Real-time systems constitute a subset of hardware and software systems in which their correctness does not only depend on the precision of the results but also on the time frame within which these results are produced [1]. These systems can further be classified into soft real-time systems, where, in the worst case scenario, missed deadlines may result in a degraded quality of service, and hard real-time systems, where all deadlines must be met constantly to avoid catastrophic failure such as loss of human life. Due to these constraints hard real-time systems are often found in safety-critical applications which frequently prompt the employment of particular scheduling approaches.

Many common operating systems use dynamic scheduling algorithms where task priorities are determined at runtime. With this approach parameters such as task arrival and execution times don't need to be entirely predictable and the inherent flexibility of such methods allows for better CPU usage, resulting in increased performance. These properties may justify the implementation of dynamic scheduling algorithms in soft real-time systems, however, the schedulability analysis, which would be required for hard real-time systems, is an NP-hard problem making the implementation and verification of such algorithms extremely difficult [2]. Consequently, hard real-time systems often resort to static scheduling methods [10]. These trade the potential performance gain and flexibility for a deterministic schedule which facilitates their

realization. Since the arrival of tasks and their worst-case execution times (WCET) are usually known beforehand for hard real-time systems, a static, deterministic schedule can easily be calculated. This ultimately yields a series of consecutive time slots, each of which have exactly one task assigned to them. This series can then be executed by the hard real-time system in a cyclic fashion without any task missing its deadline.

## 2.2. Dynamic Thermal Management

In 1965 Gordon Earle Moore, the co-founder of Intel, hypothesized that for the next decade the number of components on an integrated circuit would double every year [21]. This hypothesis was revised in 1975 by Moore himself to a doubling of components every 2 years [22]. The future of this prediction, also known as Moore's Law, is a hotly debated topic today and while it has been remarkably accurate in the past, this progress has come with major challenges [31, 17]. The high number of transistors present in a modern CPU combined with the high frequency at which they operate cause a thermal load on the processor [23], which, if left unchecked, can severely damage the CPU and other components affected by the heat dissipation. Furthermore, even if immediate damage may be avoided, high temperatures can have an adverse effect on the lifespan of the processor [33, 29] which is a significant concern for embedded real-time systems as they frequently run on the same hardware for decades (eg. automobiles and airplanes).

To alleviate the problems arising from high CPU thermal loads, hardware cooling devices are commonly employed. However, depending on their application area, their physical size and, to some extend, their cost can make them an inconvenient solution for processor overheating. Another available option (which is often used in conjunction with hardware cooling devices) is Dynamic Thermal Management. This method can be implemented purely as a software component as it tries to control the thermal load by throttling the processor accordingly. To achieve this, one of the following two approaches is taken [5].

### 2.2.1. Dynamic Power Management

If Dynamic Power Management (DPM) is employed, the CPU is transitioned between a running and sleep state. When the processor is in the sleep state its power consumption is much lower than when in an active running state. Consequently, the generated heat while in sleep mode is also diminished, giving the processor a chance to partly recover from the high temperatures reached in the running state. The challenge when using this approach is to define suitable criteria which decide when a switch between the two states should happen. The goal for these criteria is to find a satisfactory balance between peak temperature minimization and preserving performance while taking into

account factors such as the time it takes to physically switch between the states, the different environments which may affect the thermal behavior of the platform and, in the case of hard real-time systems, the requirement to meet all deadlines. Figure 2.1 shows two exemplary applications of Dynamic Power Management on a series of jobs and Figure 2.2 demonstrates their effect on the CPU temperature (these examples are of pure conceptual nature and not based on any specific algorithm).

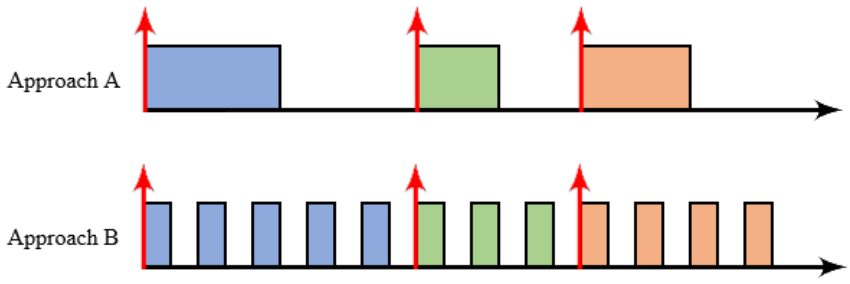



Figure 2.1.: Two Dynamic Power Management Approaches

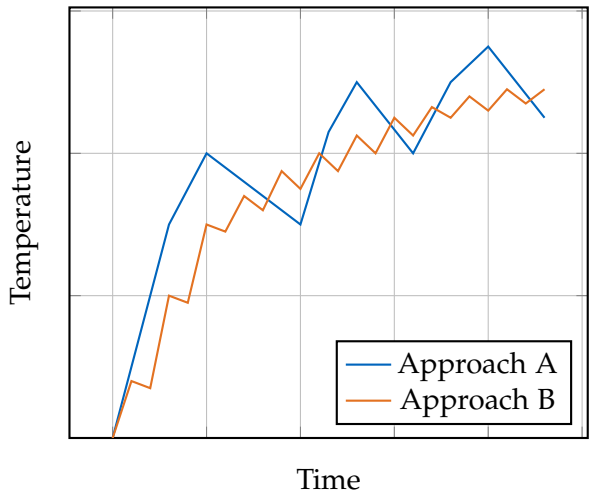



Figure 2.2.: Temperature Development for Approach A and B

**Related Work**

Extensive research has been conducted on the Dynamic Power Management approach and its possible use cases. Kumar et al. [15] investigates stop-go scheduling deriving a method called JUst Sufficient Throttling (JUST) which is able to minimize the peak temperature of the system, considering a given makespan constraint. Kumar et al. [25] furthermore studied shapers as a combination of online and offline calculations to dynamically insert idle times during execution minimizing peak temperature while guaranteeing that all jobs meet their deadlines. Cheng et al. [9] propose the Periodic Thermal Management (PTM) method to minimize the peak temperature while satisfying hard real-time constraints. Two models, serving as a trade-off between accuracy and complexity, are derived to calculate the schedule. Meisner et al. [19] introduce a Dynamic Power Management approach called PowerNap which is used to switch servers into a sleep state when idle, therefore reducing power dissipation.

### 2.2.2. Dynamic Voltage and Frequency Scaling

As an alternative to Dynamic Power Management it may also be possible to employ Dynamic Voltage and Frequency Scaling (DVFS). Instead of simply switching between sleep and active states, Dynamic Frequency and Voltage Scaling additionally allows to switch between various different active states. In each of these active states the CPU will run at a different frequency and/or voltage level (exemplary illustration in Figure 2.3, not based on any specific algorithm). This allows for the CPU to be tuned more subtly and offering more accurate control over power dissipation and thermal load. While the flexibility of this approach does provide an advantage over the Dynamic Power Management method it also comes with greater complexity, making its implementation more challenging. Compared to the binary state switch performed under DPM a more complicated model is required to derive appropriate schedules, especially when such a schedule is constrained by strict requirements as it is the case in hard real-time systems.

One common way to access this functionality is via the operating system's kernel which is used as an interface to an available driver (the CPUFreq driver). The CPUFreq driver is ultimately in charge of the frequency and voltage settings and determines what options the kernel will be able to provide. The actual policy according to which the Dynamic Voltage and Frequency Scaling is performed can be set by choosing a CPUFreq governor. The widely adopted Linux kernel implements the following governors [4]:

- **performance:** This governor runs the CPU at the maximum static frequency within preset limits. The only chance for the processor to cool down under this policy are the idle times. While these idle times may be more prevalent than

when using other governors due to the higher execution speed, ensuring a low temperature is not the primary concern of this policy.

- **powersave:** This governor sets the CPU to the lowest frequency within preset limits. This may extend the execution time of all running jobs, minimizing idle times, but due to this low frequency the power dissipation and the resulting thermal load during active running states is also diminished. From a temperature minimization standpoint, this may sound like a good idea, however, the low execution speed implicates bad performance.

- **conservative:** This governor tries to adapt the frequency setting according to the current load on the CPU. It favors low frequencies and once the processor enters an active, executing state it starts to increase the frequency in predefined steps. Once a transition into an idle state is performed, the frequency gets gradually lowered again through the various frequency levels. Therefore, the maximum frequency is only reached if the CPU is under a permanent high load. The precise functionality of this governor can be customized by setting parameters like the granularity of the frequency steps or the threshold for the frequency changes. This improves flexibility and facilitates the use of this policy across a wide range of application fields.

- **ondemand:** This governor functions in a similar way to the conservative governor as it also tries to adapt the CPU frequency to the workload, however, it does so more aggressively. As soon as the current workload surpasses a certain threshold the frequency is instantly set to its maximum possible value. Akin to the conservative governor it will gradually decrease the frequency once the CPU is idle. Overall this makes this governor more responsive to sudden high workloads while trading in potential energy efficiency. Like the conservative governor, this policy can also be customized with certain parameters, offering similar flexibility.

- **userspace:** This governor provides a more direct control over the frequency setting than the aforementioned policies. When this governor is used, an interface is provided to the root user, with which he can directly set the frequency to a specific value, as long as the frequency is supported by the given hardware. This allows for the implementation of custom policies in case none of the above satisfy one's needs (even though with considerably more overhead if implemented in the userspace [24]).

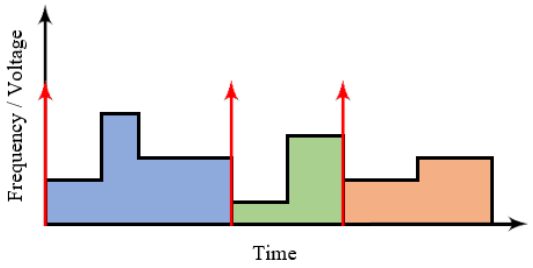



Figure 2.3.: Dynamic Frequency and/or Voltage Scaling Approach

**P-States**

A performance state (P-state) is a value pair of a frequency and a voltage defined in the Advanced Configuration and Power Interface (ACPI), an open standard that can be used by operating systems for power management of hardware devices. In the case of processors, P-states are different operating points in which the CPU can function according to different requirements such as performance, power consumption and temperature. The amount and range of available P-states is entirely dependent on the CPU, therefore, certain frequency levels might not be available for dynamic frequency and voltage scaling if the appropriate P-state is not supported by the processor.

**Related Work**

The Dynamic Voltage and Frequency Scaling approach has been widely studied by the scientific community. Sushu et al. [30] examine the performance optimization of a set of periodic tasks under thermal constraints using DVFS and furthermore proving that this problem is NP-hard. Two algorithms are presented to solve this problem, one pseudo-polynomial optimal approach and one polynomial approximated technique. Chantem et al. [6] offer an optimal solution for work maximization considering thermal constraints applicable for general-purpose uniprocessor systems. Additionally the assertion is made that the optimal speed schedule solution must be periodic. Chen et al. [8] study thermal constrained scheduling for real-time tasks in a DVFS system. Two methods are derived to either minimize response time under a thermal constraint or to minimize the temperature under a timing constraint. Dhiman et al. [11] propose a DVFS technique for multitasking systems which employs an online learning algorithm and the available runtime statistics to determine which voltage and frequency should be used in any given moment. Le Sueur et al. [16] investigates the feasibility of DVFS approaches for modern processors. It is concluded that the effectiveness of voltage and

frequency scaling has been reduced in modern systems due to the high static power consumption, smaller dynamic power range and increasingly efficient sleep modes.

## 2.3. Genetic Algorithm

Genetic algorithms are an approach to optimization problems based on natural selection [13]. Much like through evolution in nature, genetic algorithms attempt to improve given solutions to a problem by progressing through several generations of these solutions. At the outset a population of possible solutions (the first generation) is given, large enough in size so that suitable diversity within this population can be ensured. Starting with this population, the genetic algorithm begins iterating over two main steps to recreate the process of natural selection:

1. **Selection:** In this first step the current population of solutions is evaluated according to a fitness function. This function serves as a criteria to judge the quality of every individual in the current generation. Based on this fitness, a subset of individuals is chosen as the basis for the next generation. Less fit individuals may also be included in a future generation in order to keep high diversity and avoid premature convergence toward a mediocre solution [14].

2. **Reproduction:** Based on the individuals selected during the first phase, these solutions are now designated parents which serve as the foundation for the new generation of solutions. The new child individuals are created through combinations of their parent's properties and potential mutation of some of their features.

This procedure is designed to edge ever closer to the optimal solution by picking individuals which are near the optimum. This is done until a termination criteria, such as a satisfactory individual or plateauing of fitness, is met.

Genetic algorithms are commonly employed in situations where a clear solution to a problem is unknown and not easily derivable. Problems with a high number of unknown variables lend themselves particularly well to this approach since mathematical solutions to such problems can be very difficult to find, whereas genetic algorithms seemingly find solutions "on their own" by trial and error.

### 2.3.1. GMPT

The elemental features of genetic algorithms such as the large population size and the many iterative developments of each individual make it highly impractical to evaluate the performance of every solution experimentally. This calls for a model which can

be used to approximate the fitness of each individual. GMPT resorts to a thermal model that is commonly employed in related research [7, 20, 15]. Here the temperature development of the CPU over time is described by

$$T(t) = \frac{A_m}{B_m} + (T(t_0) - \frac{A_m}{B_m})\, e^{-B_m(t-t_0)}, \tag{2.1}$$

with $T(t_0)$ being the initial processor temperature and $A_m$ and $B_m$ being factors which describe the CPU peak temperature given by $T_m^\infty = \frac{A_m}{B_m}$, i.e. the temperature $T(t)$ reached for $t \to \infty$. $A_m$ and $B_m$ are derived from the processor's power consumption (leakage and dynamic) and therefore determine the thermal behavior of the platform for a specific operating mode. Consequently every CPU operating mode has exactly one constant $T_m^\infty$ associated with them.

While this equation will be used for the evaluation of the experiments, it cannot directly be applied to schedules which run multiple different frequencies. Therefore an additional transformation is performed by using

$$T_{peak}^{PTM}(q) = max(T_1^{st}, \ldots, T_i^{st}, \ldots, T_q^{st}).$$

The Periodic Thermal Management (PTM) schedule that is returned for each individual by GMPT consists of a period partitioned into various segments, each of which have their own frequency/voltage level assigned to them. By running this schedule, the CPU will eventually reach a state where the temperature at the end of a segment won't change from one period to the next anymore (see Figure 2.5). This temperature is designated as the steady temperature $T_i^{st}$. $T_{peak}^{PTM}(q)$ is therefore an upper bound for the temperature which is dictated by the highest frequency/voltage setting within a schedule. This temperature limit is ultimately used to assess the fitness of every schedule via

$$fitness = \frac{1}{T_{peak}^{PTM}(q)}.$$

On the basis of this $fitness$ value, individuals are selected and used to generate new schedules. If these new individuals still fulfill the hard real-time constraints they are used as part of the next generation of schedules.

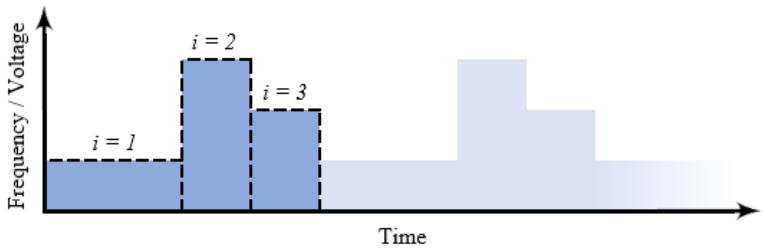



Figure 2.4.: One possible PTM schedule for $q = 3$

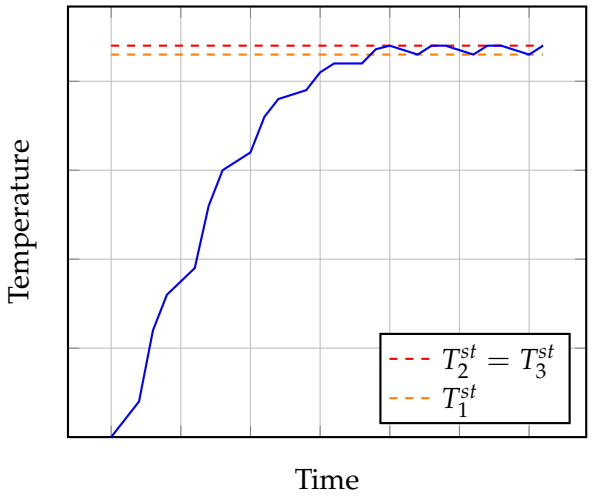



Figure 2.5.: Possible temperature development for schedule depicted in Figure 2.4

### 2.3.2. Boxplots

Boxplots are a graphical representation of groups of numerical data. A boxplot consists of a box which symbolizes the central 50% of the data points, the distance between the lower and upper bound of this box is also known as the interquartile range (IQR). Within this box a line marks the median of the entire dataset. Extending out from the upper and lower bound of the box are the so called whiskers. These whiskers extend to the data points furthest away from the box up to a maximum length of 1.5x IQR (based on Tukey boxplot). All data points which lie further away are considered outliers.

The actual values of the measured parameters in this thesis are not known, but the the measured data points are expected to be distributed normally (or at least closely resemble such a distribution). In this case the 1.5x IQR limit represents the range of

values within which 99% of all measurements are expected to lie.

Boxplots lend themselves especially well to display the kind of experimental data that has been recorded during this thesis. They can give a simple but clear overview of how the recorded data is distributed an provide a strong indication of where the true value of the measured parameter lies.

# 3. Approach

This chapter provides insight into the approach taken to run the schedule experiments. First, the framework used for the experiments is introduced, giving an overview of its functionality and of the required modifications performed on the framework itself and on the platforms upon which it was employed. Then, a preliminary examination of the two platforms is performed, including a study and analysis of their thermal behavior. This serves as a basis for the upcoming schedule experiments.

## 3.1. Framework

The evaluation of the various schedules generated by the GMPT algorithm requires a highly controlled environment. Since the temperature measurements of the processor are of paramount importance when comparing the results of the different algorithms, it needs to be ensured that, while the schedules are running, interference by other unrelated processes is minimal. Therefore, a special software setup is required. While open-source frameworks which were specifically designed to prototype real-time scheduling algorithms are available, they do not fully support the required functionality (such as custom frequency scaling). This has led to the implementation of a new framework by L. Bersentes [3] which provides the required features and will be used within the scope of this thesis.

### 3.1.1. Architecture

The framework is implemented as a series of threads each of which represent one component within the process management system and functions as a full-scale simulation of it. Figure 3.1 illustrates the Architecture of the framework.
The main building blocks, each of which run in their own thread, are:

- **Main:** This component offers all the basic functionality such as initialization, reading input parameters as well as the creation of all other threads. Once all threads have started the simulation of the given schedule begins and this component sleeps throughout the entire experiment to minimize its performance impact. It is therefore completely inactive for the duration of the simulation.

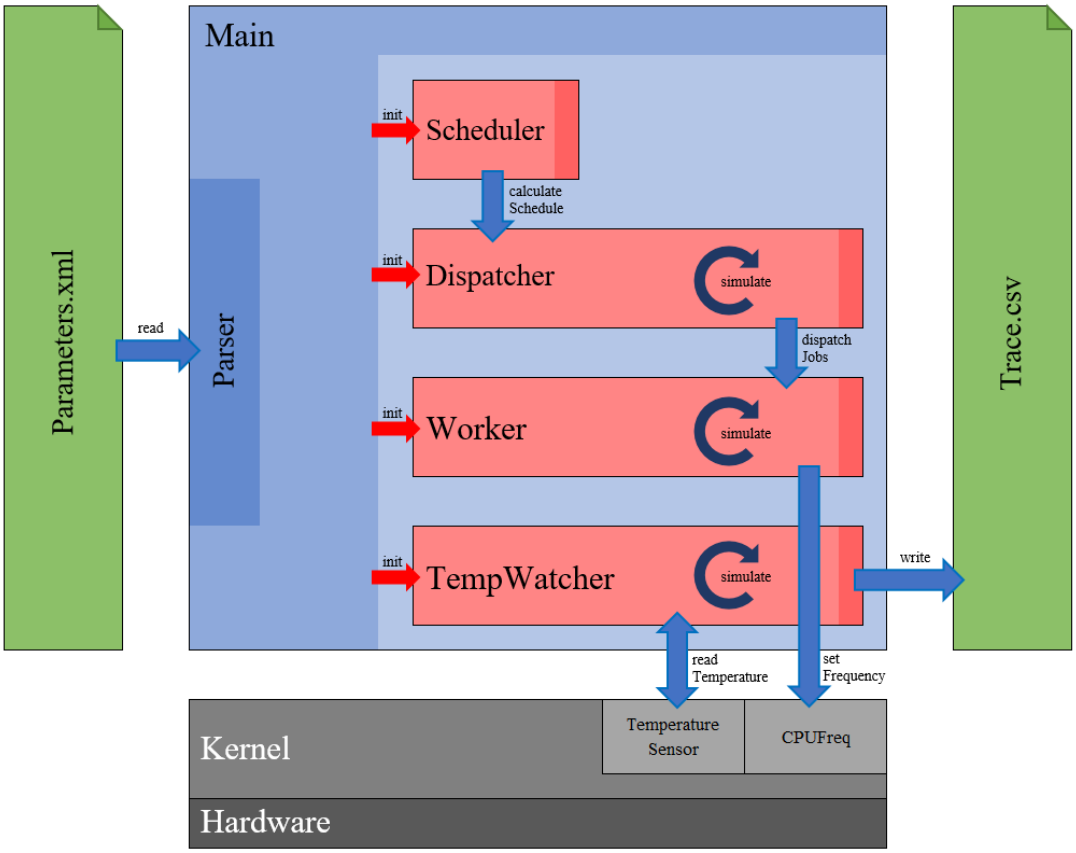



Figure 3.1.: Framework Architecture

- **Scheduler:** This component is responsible for scheduling decisions. As it currently only supports static scheduling approaches, all scheduling decisions can be determined beforehand. This results in a minimal overhead as the thread is able to exit almost instantly, finishing much earlier than the actual simulation.
- **Dispatcher:** This component is in charge of triggering all the jobs at the appropriate moment in time. Unlike in the case of the scheduler, the dispatcher needs to be active throughout the simulation as it needs to release the jobs by communicating with the worker. Even though all job parameters are known prior to execution it needs to be ensured that these parameters are actually enforced, otherwise there would be no guarantee that constraints such as period length apply.
- **Worker:** This component is used to manage all job related functions and rep-

resents one CPU core. It's primary purpose is to execute the jobs released by the dispatcher and to do so by forcing the highest possible workload on a CPU core. It is furthermore responsible for switching between idle and active states (when idle states are part of the schedule) and for changing to the different CPU frequencies within one period.

- **TempWatcher:** This component is used to monitor the CPU temperature throughout the entire simulation. The functionality has its own thread assigned to it to ensure that these measurements can be performed regularly and independently of the current state of the other components.

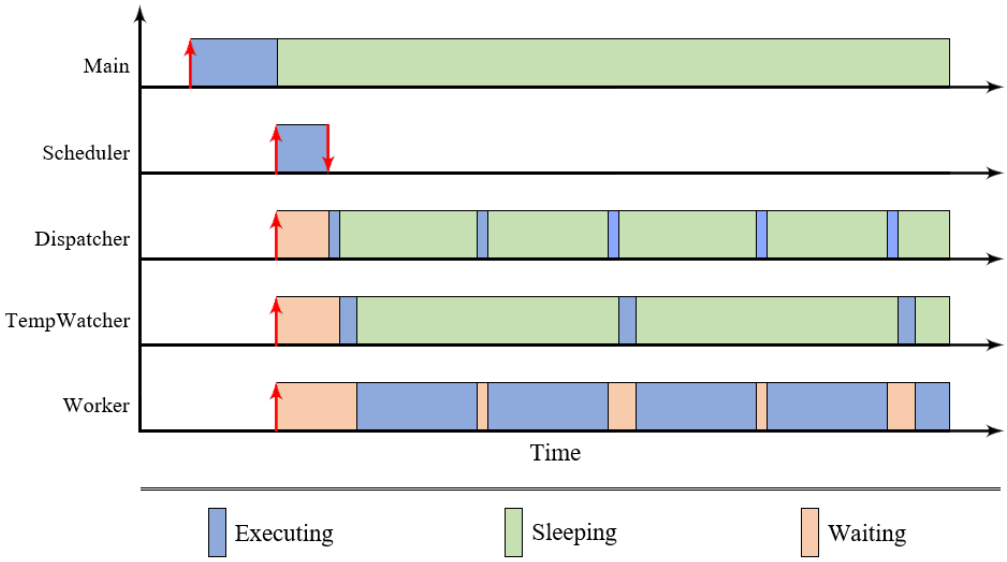



Figure 3.2.: Framework Thread Schedule

An exemplary execution of these main components on a single core can be seen in Figure 3.2 (the threads are sorted by their priority, Main having the highest priority).

### 3.1.2. Platform Adjustments

The GMPT algorithm will be evaluated on two different hardware platforms to analyze its feasibility. While the framework can run entirely inside the userspace, which implies high portability, it still relies on certain functionality provided by the kernel. Consequently some changes to the framework still have to be made when employing this framework in heterogenous environments.

**Platform: Notebook**

The first hardware platform upon which the GMPT algorithm will be tested is the LENOVO IdeaPad U330p operating an Intel Core i5-4210U and running Ubuntu 14.04 LTS.

The very first step is to disable the integrated Intel P-State driver which does not provide control over the userspace governor and therefore would not allow to implement the custom frequency scaling that is required. Additionally the location of the kernel interfaces needs to be adapted and the available settable frequencies need to be added to the framework.

A common way to set the CPU frequencies is via shell commands and this is also approach that has been implemented in the framework. While this approach may be simple and easy to implement, measurements (Figure 3.3) have shown that the execution of this command take a considerable amount of time (at times more than 1 ms) which may stall the execution of the job themselves therefore affecting the temperature development. This is especially problematic since the schedule provided by GMPT may only require the CPU to run at a certain frequency for 1 ms. The time it takes for the CPU to physically switch between frequencies is reported to be within the tens of microseconds [12, 28, 27], arguably negligible compared to the time spent executing the command. The outlined problem has called for an improvement of this approach.

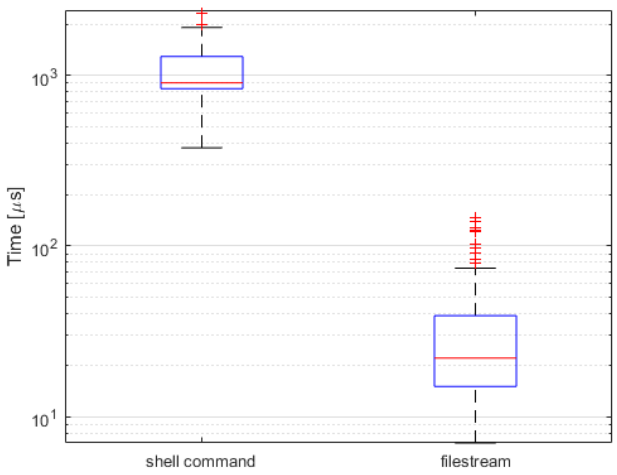



Figure 3.3.: Execution time of the two frequency switch implementations

Still using the same kernel interface, instead of resorting to shell commands, I chose

to communicate with it directly by the filestreams available in C++. As Figure 3.3 illustrates, the performance was increased by more than an entire order of magnitude, which should result in better confidence in the results.

#### Platform: Raspberry Pi 3B

To further improve the trust in the experimental results, the GMPT algorithm should also calculate a schedule for the Raspberry Pi 3B. This platform employs a 1.2 GHz 64-bit quad-core ARMv8 CPU integrated within its BCM2837 SoC. Raspbian Jessie Lite was chosen as the OS as it is expected to have the least impact on the generated results and overall performance due to its minimal image.

The Raspberry Pi only supports one CPUFreq driver and while it does support the userspace governor, the only two frequency levels that can be set are the minimum (600MHz) and maximum (1.2GHz) frequency. This is not sufficient for our use-case because at least four equidistant levels are needed. The first possibility to be investigated was expanding the userspace governor to support more frequency levels by adding the functionality in the kernel. Once this new customized kernel was tested, it turned out that the functionality could not be added to it due to missing firmware support [26]. As an alternative it was discovered that the CPU frequency could be set by the vcmailbox driver. This driver is responsible for the communication between the processor and the VideoCore and provides an interface which can be used for the purpose of this thesis [18]. In light of the attempted kernel customization it was noticed that the frequency reported by the kernel was merely based on the governor settings and hence not measured. Due to this implementation incorrect frequencies were reported by the kernel when set using the vcmailbox driver. This led to the employment of vcgencmd (a function provided by the VideoCore firmware [32]) which determines the true CPU frequency via clock measurements. The tool further led to the verification that vcmailbox can set permanent frequency levels regardless of current CPU workload or active governor, overriding any influence it may have.

### 3.1.3. Overhead Analysis

To be able to accurately interpret the experimental results, the overhead introduced by the framework itself needs to be considered. Due to disruptions by other threads and by certain instructions like the frequency change, the Worker won't be able to be active for the entire experiment. This overhead can be attributed to three main factors:

- **Job Dispatching:** This is expected to have the least overall impact since it simply manages a list of jobs. The Dispatcher runs in its own thread and therefore is still considered in this analysis.

- **Temperature Measurements:** These are heavily platform dependent since the speed at which the measurements can be performed mostly depends on the built-in sensor.
- **Frequency Changes:** As outlined in subsection 3.1.2, the performance impact of the frequency switches is mostly implementation dependent. The adjustments made to the framework regarding this procedure are taken into account in this analysis.

Based on a 60 second experiment, the overhead introduced by these influences has been recorded. The average time consumed by single instances of such interruptions can be seen in Table 3.1.

Table 3.1.: Average overhead introduced by the framework

| | Notebook | Raspberry Pi 3B |
|---|---|---|
| Avg. Job Dispatch [$\mu s$] | 1.38 | 16.37 |
| Avg. Temp. Check [$\mu s$] | 7221.45 | 587.02 |
| Avg. Freq. Change [$\mu s$] | 27.64 | 6308.34 |

With the default settings of the framework (under which all experiments were run), the Dispatcher runs ten times per second. Considering the measured times, in a one minute experiment this amounts to approximately 830 $\mu s$ ( 0.001% of 60s) on the Notebook and 9820 $\mu s$ ( 0.01% of 60s) on the Raspberry Pi. This is not expected to have a major impact on the validity of the results.

The temperature is measured five times per second (with default framework settings). In a 60 second experiment this results in overall 2.2 s ( 3.6% of 60s) on the Notebook and 176 ms ( 0.3% of 60s) on the Raspberry Pi. On both platforms this has a considerably bigger impact than the Dispatcher. However, this task has been assigned to a different core and should therefore not interrupt the Worker thread, minimizing its impact.

The frequency switches show the biggest discrepancy between the two platforms. The actual impact this has on the overall execution heavily depends on the schedule used. If a schedule performs many frequency switches, the effect will be much more pronounced. On the Notebook this is expected to be less of an issue since the switch itself is performed fairly quickly, on the Raspberry Pi, however, this can severely impact the results due to the large amount of time dedicated to switching frequencies. The current implementation on the Raspberry Pi relies heavily on the driver provided by the OS (see subsection 3.1.2) and steps taken to alleviate this problem have been mostly unsuccessful. This may be especially problematic if the timeslots within a schedule

are very short and it needs to be taken into account when evaluating the results of future experiments on the Raspberry Pi (this does not affect the thermal profiles, as no frequency changes are performed there).

## 3.2. Experiments

The goal of the experiments is to provide thermal measurements for various schedules. This data is subsequently used to analyze different algorithms and compare them side-by-side.

The framework that would ultimately be used to run the schedules was chosen on the basis of its high portability and due to the required functionality that was already implemented in it. For a first platform to test the framework and simulate the schedules, a notebook was chosen as it was readily available, was running a Linux distribution and did not have much additional software installed on it which could disrupt a smooth schedule simulation. As a second platform to run and verify the schedules, the Raspberry Pi 3B was chosen. Here the framework could easily be run without any interference potentially caused by fans or heat sinks while still providing the commodity of an ordinary operating system. This lightweight OS also promised to generate easily reproducible and reliable temperature measurements due to its lack of a graphical user interface and consequently the lack of background calculations needed to support it.

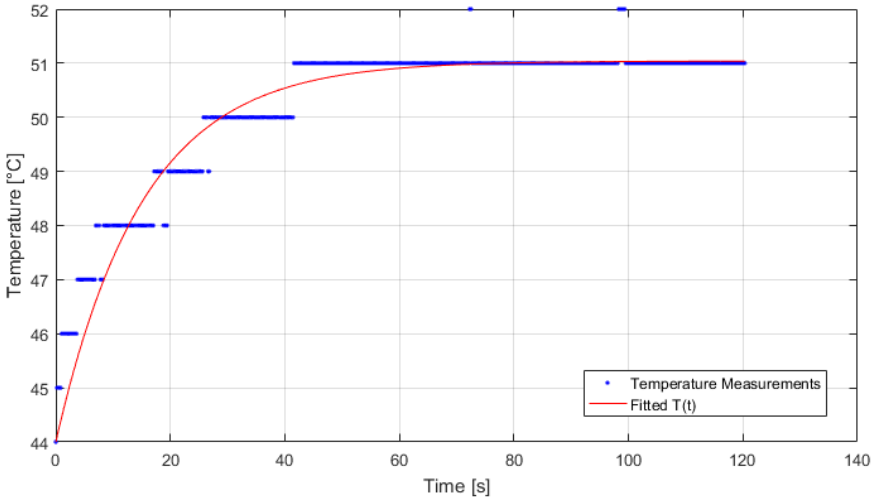



Figure 3.4.: T(t) fitted to temperature measurements

The goal of all evaluated algorithms is the peak temperature minimization within hard real-time constraints. The highest measured temperature may not be an accurate representation of the peak temperature as it may be too low due to unexpected interference or it simply has not yet been reached by the end of the experiment. Preliminary test runs have shown that for both platforms and most schedules $T_m^\infty$ is reached after about 60 seconds, hence all experiments ran for this duration, however, a better estimate of the peak temperature would still be desirable. Consequently the model presented in 2.3.1 is employed by fitting the function (2.1) to the temperature measurements (Figure 3.4), this should return a more precise value for $T_m^\infty$. To further improve confidence in the results and detect possible outliers which may sabotage the data integrity, every experiment was run 10 times. In addition to these precautionary measures all of the experiments were performed in the same controlled environment under a constant ambient temperature. During the simulations, the platforms were dedicated entirely to the framework and no other work was performed on them.

### 3.2.1. Platform Frequency Ranges

The two platforms used for the experiments do not provide the same flexibility in their frequency settings. This and the need to limit the possible options available to the GMPT algorithm (to avoid overwhelming complexity) has led to the decision of constraining the settable frequencies to 4 equidistant levels denoted as 0.4, 0.6, 0.8, 1 and a 5th operating mode 0 which represents the idle state.

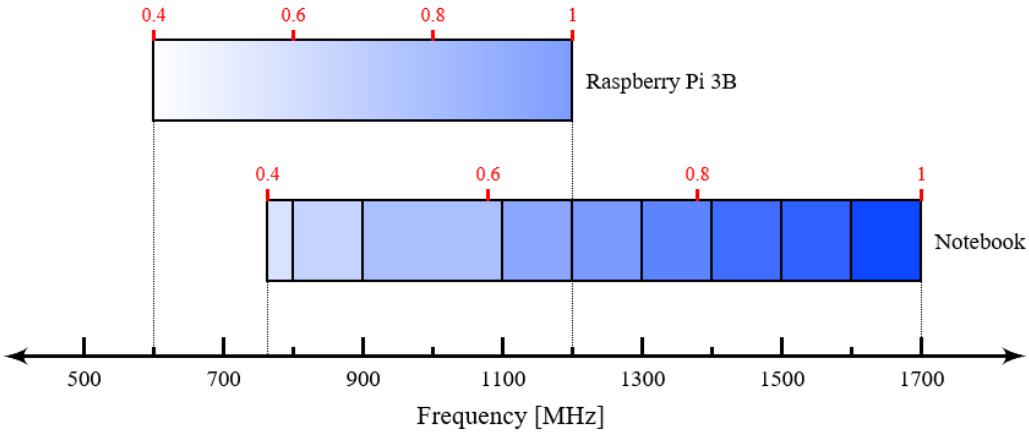



Figure 3.5.: Frequency Ranges of the two hardware platforms

Figure 3.5 provides a comparison of the two frequency ranges available for the employed platforms. The first observation that can be made is that the notebook has a wider frequency range available. Even though the smallest possible frequency does not reach as low as the Raspberry Pi's range, this allows for more overall flexibility. Consequently the notebook has the ability to react more aggressively in certain situations. This possibility for drastic changes in the current frequency can be a great tool to influence the temperature development more swiftly and responsively, however, it also results in an increased performance variance which may be challenging to handle in time constrained systems. It should further be noted that only a very specific set of discrete frequency steps are available as settings. This leads to the problem that the frequency steps denoted as 0.6 and 0.8 may not be available as actual settings (as it is the case for the notebook platform). In such a case the frequency closest to the desired setting is chosen by the framework. In contrast to this, the mailbox driver employed on the Raspberry Pi allows to set a very large amount of frequency levels. Here the frequency can be specified down to the kHz level. This allows for very accurate frequency adjustments within this overall smaller range. In this current use-case such accuracy is not really required and would actually be highly problematic if used to its full extend. Apart from rendering the schedule optimization very costly, the large amount of available frequency options could also entail a high number of frequency switches resulting in a very large overhead used entirely for the switching process. Therefore the schedules for this platform have also been restricted to use four frequency levels. On the Raspberry Pi these frequency settings are precisely available due to the given step granularity and don't need to be approximated like on the notebook.

### 3.2.2. Thermal Profiles

Before the GMPT algorithm is able to calculate any schedules for a given platform, thermal profiles of that platform have to be provided. These profiles give an insight on how temperature development is affected by a specific frequency setting. This information is crucial to understanding how the processor will behave for a certain schedule and is therefore important when making according predictions to optimize it.

These thermal profiles were determined by running schedules inside the framework at constant frequency settings. The first schedule (0) consisted of idle times only, here the frequency was set to the minimum and no calculation were performed during any period (i.e. the CPU was completely idle for the entire simulation). For the following simulations the four equidistant frequency settings 0.4, 0.6, 0.8 and 1 were used.

**Thermal Profile: Notebook**

Additionally to the adjustments detailed in 3.1.2 the built-in fan was disabled to get more accurate and reliable readings regarding the temperature dissipation. Other hardware cooling devices (such as the heat sink) were not removed in order not to corrupt the platform integrity. Since the focus lied on the performance of different scheduling algorithms on a single core, all unused cores were constantly set to their lowest possible frequency and kept at idle.

Table 3.2.: Average results for notebook platform

| Frequency Setting | Average $T_m^\infty$ [°C] | Average $R^2$ |
|---|---|---|
| 0 | 43.933 | - |
| 0.4 | 47.183 | 0.721 |
| 0.6 | 48.895 | 0.812 |
| 0.8 | 51.322 | 0.900 |
| 1 | 65.680 | 0.860 |

The average results for $T_m^\infty$ can be seen in Table 3.2 together with their average coefficient of determination $R^2$ which serves as an indicator of how closely the data matches our given model. $R^2$ is not reported for the idle measurements since it does not apply here (the idle temperature is constant).

Figure 3.6 displays the thermal profile for the notebook platform including the results for idle and the four frequency settings with maximum workload. This analysis shows a considerable jump in temperature for the highest frequency level. This can be explained by the relation [12]

$$P = CV^2F,$$

where $P$ is the power dissipation, $C$ is the capacitance, $V$ is the voltage and $F$ is the frequency at which the integrated circuit is operating. While the frequency on its own affects the power dissipation linearly, the CPU implicitly adapts the voltage accordingly (higher frequency also means higher voltage). Therefore, for each experiment, voltage and frequency were increased, which, when combined, result in a cubic influence on power dissipation, ultimately leading to the cubic behavior of the temperature development. The data provided for the idle run should not be considered as a part of this function since it differs in CPU utilization from the other four settings.

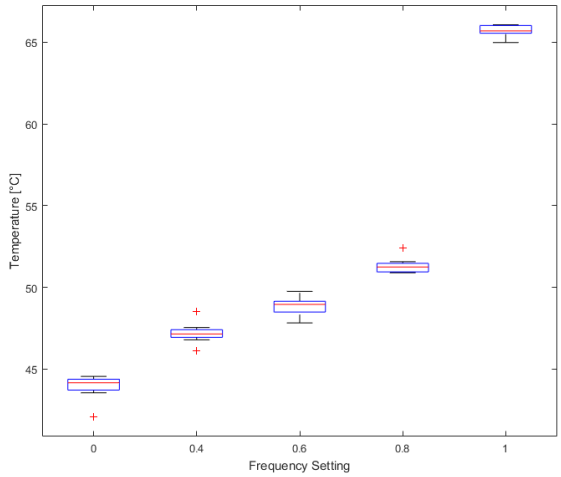



Figure 3.6.: Thermal profile of the notebook platform

**Thermal Profile: Raspberry Pi 3B**

Unlike in the case of the first platform, hardware cooling devices like fans and heatsinks are not preinstalled on the Raspberry Pi 3B. The measurable results were therefore expected to be a better representation of the actual processor's thermal performance. The CPU on this platform includes four cores but the mailbox driver used to set the frequencies can only set the frequencies of all cores at once. The frequency scaling therefore also affected the other three cores but since they were idle only a small impact on the overall thermal behavior was expected.

The average results for the Raspberry Pi platform and the associated $R^2$ values are reported in Table 3.3.

Table 3.3.: Average results for Raspberry Pi 3B

| Frequency Setting | Average $T_m^\infty$ [°C] | Average $R^2$ |
|---|---|---|
| 0 | 45.087 | - |
| 0.4 | 49.198 | 0.811 |
| 0.6 | 50.255 | 0.797 |
| 0.8 | 51.965 | 0.914 |
| 1 | 52.792 | 0.933 |

The profile for the Raspberry Pi depicted in Figure 3.7 shows a clear difference from the notebook profile. The maximum frequency does not cause the big temperature increase observed on our first platform. In order to exclude possible interference caused by other cores the experiments were run a second time with three of the four cores completely turned off, however this resulted in a temperature increase of approximately 6°C for every frequency setting (keeping the relative temperature changes constant). An evaluation of the CPU utilization returned a workload of >97% throughout the simulation, which is identical to the notebook data. Extending the simulation time also yielded no significant changes in the thermal profile. Since the temperature changes for similar frequencies were also small on the first platform it was ultimately concluded that the lack of major temperature changes is the result of a smaller frequency range and a lower available maximum frequency. This can not be fully verified due to the lack of data such as the thermal design power specification of the ARM Cortex-A53 (datasheet not publicly available). While this smaller temperature range may render the comparison of different approaches more challenging, a smaller spread of the experimental results within every frequency setting can be observed. This implies a more accurate temperature control via the frequency setting and is expected to improve overall data quality for the upcoming algorithm tests.

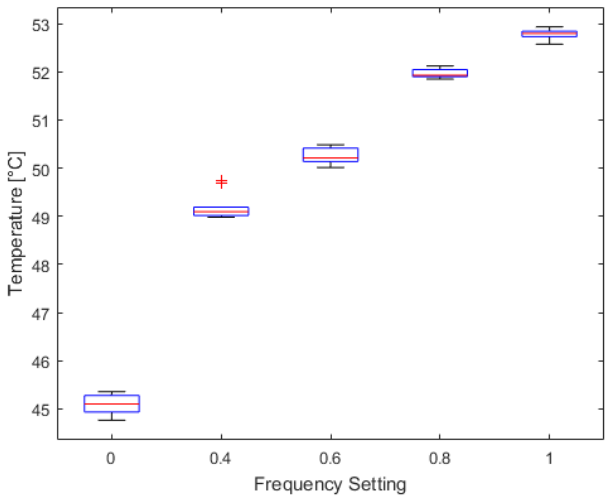



Figure 3.7.: Thermal profile of the Raspberry Pi 3B

# 4. Results

Based on the thermal profiles which were determined experimentally in 3.2 the GMPT algorithm will be used to generate schedules for different workloads. Then, for those same workloads, schedules are also derived by the PMPT algorithm [9]. These are DPM schedules that also satisfy hard real-time constraints and were calculated by brute force to find the best ratio between the active and inactive CPU state. This produces very accurate data but also requires lengthy computations, resulting in a large overhead. The two approaches will be employed twice, once for a single-task model and once for a multi-task model. The data returned by these experiments will serve to:

(i) analyze the feasibility of the GMPT algorithm by assessing the performance of its schedules in an environment that closely resembles a typical embedded real-time system.

(ii) compare the GMPT schedules to another state of the art approach by examining its effect on the thermal development of the processor.

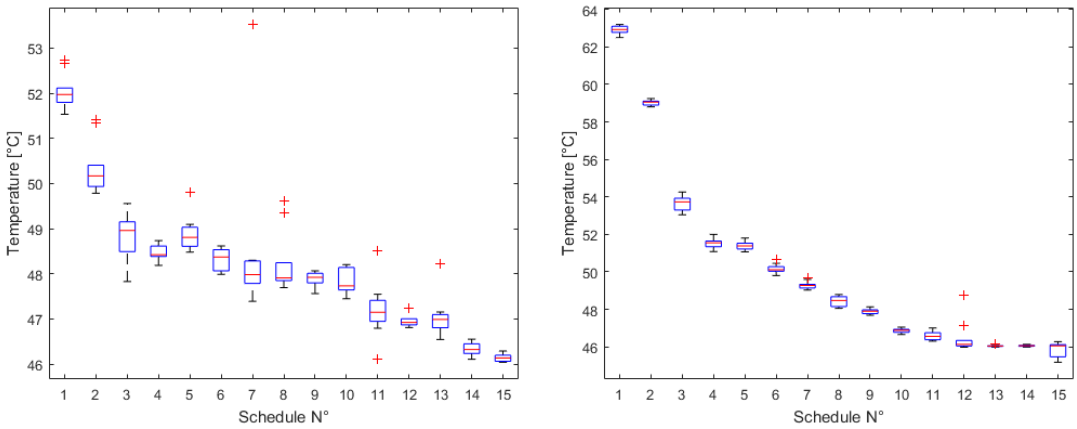
## 4.1. Platform: Notebook

Two sets of experiments were run on the notebook platform. The first set served to compare the solutions proposed by the two methods within a single-task simulation. In the second set the algorithms tackled a multi-task simulation.

### 4.1.1. Single-Task

The schedules generated by the two algorithms for 15 different workloads can be seen in Table A.1 (Appendix). This table clearly demonstrates that the increased effort that is invested in the computation of DVFS schedules doesn't necessarily translate into a more complex schedule. In many cases the schedule derived by GMPT contains only two frequency switches which indicates that the overhead for such schedules won't be much larger than for PMPT schedules (if the frequency setting of the first and last timeslot within a GMPT schedule match, no switch is required between the periods). The thermal behavior of the processor for the GMPT and PMPT schedules is displayed

in Figure 4.1.




(a) Results GMPT (b) Results PMPT

Figure 4.1.: Thermal behavior of schedules in single-task environment

The first few schedules show how a high workload can drastically affect the CPU temperature. Overall both approaches display a similar pattern, naturally struggling with the very high workloads and eventually stabilizing at lower workloads.

The biggest performance gap between GMPT and PMPT solutions are the high workloads (Table 4.1). This can be attributed to the cubic influence the frequency/voltage has on the processor power dissipation. Being able to lower the frequency and voltage for a long time period, even if just by a small amount, can have a major impact on the thermal behavior of the platform especially in high frequency ranges. This primarily affects high workloads since in these cases the CPU runs at the maximum frequency for extended periods. This effect is also reflected in the previously computed thermal profile of the notebook.

Table 4.1.: Median peak temperature for all workloads and their performance gap

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| GMPT | 51.9 | 50.1 | 48.9 | 48.4 | 48.8 | 48.3 | 47.9 | 47.9 | 47.9 | 47.7 | 47.1 | 46.9 | 46.9 | 46.3 | 46.1 |
| PMPT | 62.9 | 59 | 53.7 | 51.5 | 51.3 | 50.1 | 49.2 | 48.4 | 47.8 | 46.8 | 46.5 | 46.1 | 46 | 46 | 46 |
| P.Gap | -11 | -8.9 | -4.8 | -3.1 | -2.5 | -1.8 | -1.3 | -0.5 | 0.1 | 0.9 | 0.6 | 0.8 | 0.9 | 0.3 | 0.1 |

Toward lower workloads the difference between the two approaches becomes less

clear with the GMPT schedules occasionally resulting in higher peak temperatures. This can, at least in part, be attributed to the increased effectiveness of the idle times due to their longer duration for low workloads. Additionally, with decreasing workloads the peak temperature approaches and converges toward the idle temperature. The performance gap between the two approaches will therefore inevitably shrink. Concerning the accuracy of the measurements Figure 4.2 demonstrates that the spread of data points is very similar for both approaches and that the mean peak temperatures for GMPT and PMPT at low workloads often lie within the spread of each others data points. The performance gap at these workload levels is therefore not considered significant.

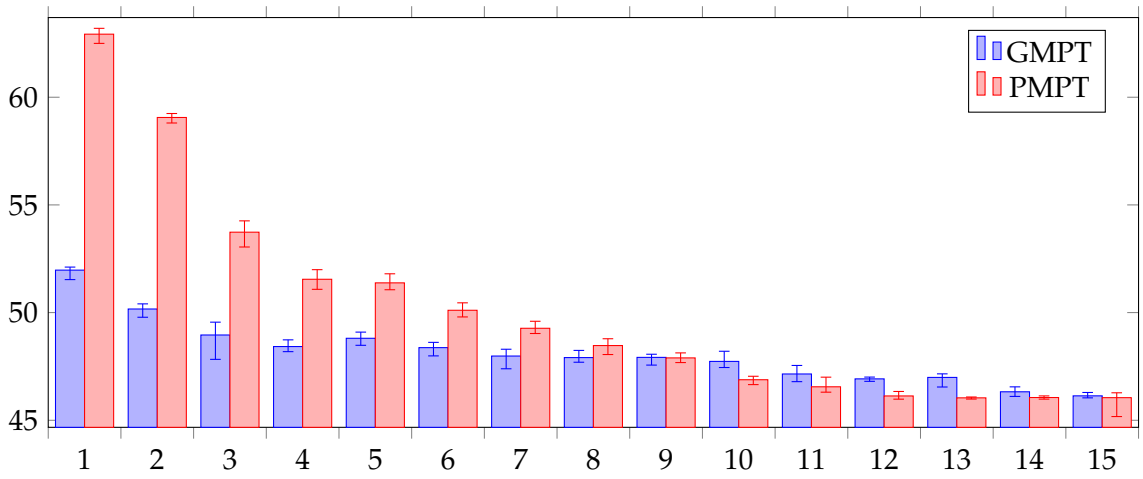
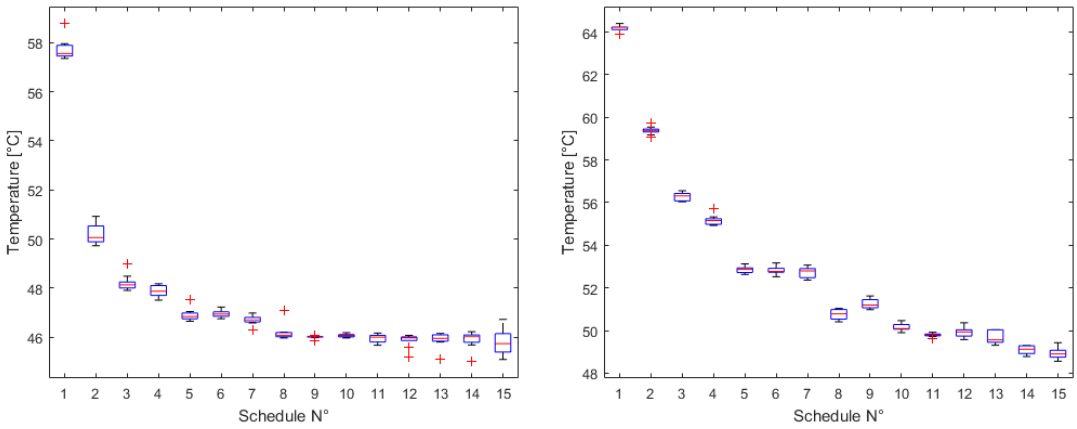



Figure 4.2.: Comparison[1] of the temperature measurements for GMPT and PMPT

### 4.1.2. Multi-Task

The comparison within the multi-task simulation is also based on 15 schedules designed for different workloads, they are reported in Table A.2 (Appendix). Similarly to the previous comparison, many of the GMPT schedules do not require any more frequency switches than the PMPT schedules, diminishing the overhead. Figure 4.3 depicts the thermal reaction of the notebook platform for the new schedules generated by GMPT and PMPT.

[1] The bar values correspond to the median and the error bars correspond to the lower and upper inner fences displayed in the boxplots in Figure 4.1




(a) Results GMPT (b) Results PMPT

Figure 4.3.: Thermal behavior of schedules in multi-task environment

The overall thermal pattern for both scheduling approaches is again similar, however, contrary to the first comparison, the peak temperature reached by the DPM schedules seem to converge toward a higher temperature level (or at least converge significantly slower). The performance gap for very high workloads is not as big as for the prior comparison but for very low workloads a considerable difference between the two approaches is still visible (Table 4.2).

Table 4.2.: Median peak temperature for all workloads and their performance gap

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| GMPT | 57.5 | 50 | 48.1 | 47.8 | 46.8 | 46.9 | 46.7 | 46 | 46 | 46 | 46 | 45.9 | 45.9 | 46 | 45.7 |
| PMPT | 64.2 | 59.3 | 56.3 | 55.1 | 52.8 | 52.7 | 52.7 | 50.7 | 51.1 | 50 | 49.8 | 49.9 | 49.5 | 49.1 | 48.9 |
| P.Gap | -6.7 | -9.3 | -8.2 | -7.3 | -6 | -5.8 | -6 | -4.7 | -5.1 | -4 | -3.8 | -4 | -3.6 | -3.1 | -3.2 |

The idle temperature of the platform obviously doesn't change depending on the amount of tasks. As visualized in Figure 4.4, the spread of the data points is also too tight to attribute this discrepancy to pure measurement inaccuracies. This phenomenon is probably mostly owed to the fact that a schedule consisting of multiple tasks cannot always be condensed down into one continuous execution block if all tasks run at maximum speed. This means that PMPT must ensure that the active times are sufficiently large to accommodate all tasks. This could lead to situations where during some of the active time, no tasks are running (as they are either finished, haven't been started or cannot start yet). For the GMPT approach this does not constitute a problem since the

execution times of each tasks can be "stretched" so that every task finishes as closely as possible to the start of the next task. Under these circumstances GMPT clearly provides the better results from a pure peak temperature minimization standpoint.

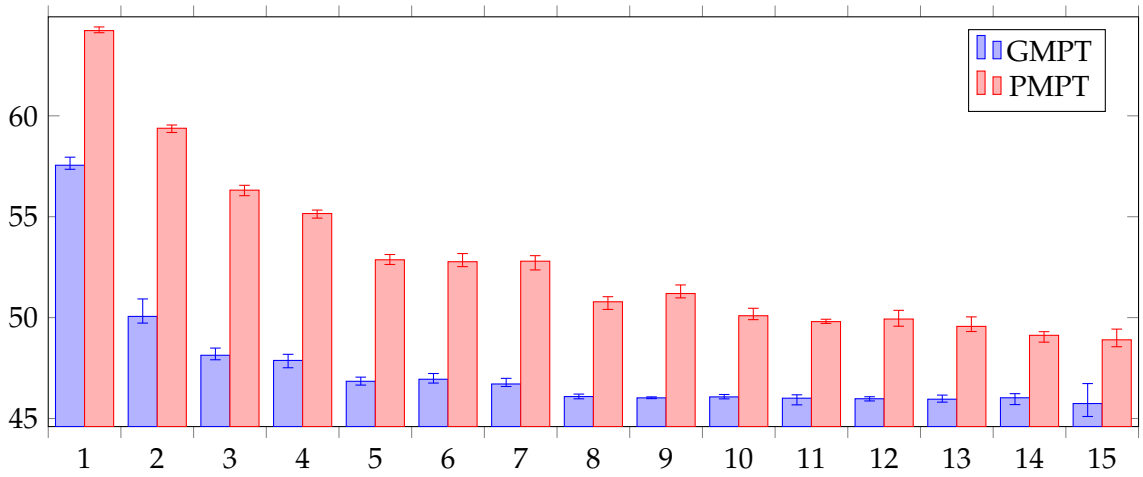



Figure 4.4.: Comparison[2]of the temperature measurements for GMPT and PMPT

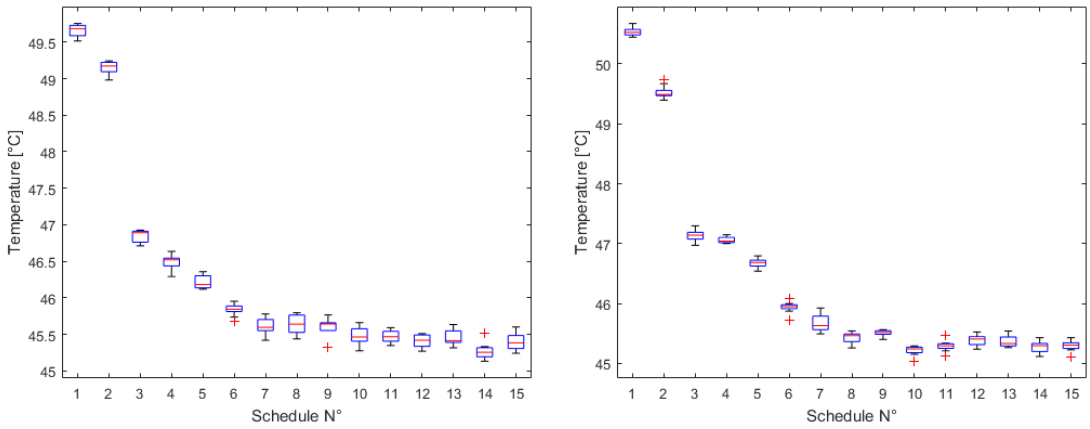
## 4.2. Platform: Raspberry Pi 3B

On the Raspberry Pi platform the GMPT and PMPT algorithms were again applied to calculate schedules for single- and multi-task simulations.

### 4.2.1. Single-Task

The schedules generated by GMPT and PMPT for the different workloads in the single-task environment can be seen in Table A.3 (Appendix). Even though it is not as pronounced as on the notebook platform, some of the schedules that were computed by the genetic algorithm also require only few frequency changes. For these schedules the influence of the overhead is expected to be very similar for both approaches. However, overall the schedules for the Raspberry Pi do seem to require more frequency changes than those on used on the Notebook. This can at least in part be attributed to the smaller temperature range available on this platform, which likely makes it more difficult to determine which frequency is best at any given point in time.

[2]The bar values correspond to the median and the error bars correspond to the lower and upper inner fences displayed in the boxplots in Figure 4.3




(a) Results GMPT (b) Results PMPT

Figure 4.5.: Thermal behavior of schedules in single-task environment

Figure 4.5 displays the thermal behavior of the Raspberry Pi CPU for the two kinds of schedules. As indicated by the thermal profiles, the changes in temperature are not as large as on the notebook platform but the temperature development through the 15 schedules still resembles a similar pattern.

Table 4.3.: Median peak temperature for all workloads and their performance gap

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| GMPT | 49,7 | 49,2 | 46,9 | 46,5 | 46,2 | 45,8 | 45,6 | 45,6 | 45,6 | 45,5 | 45,5 | 45,4 | 45,4 | 45,3 | 45,4 |
| PMPT | 50,5 | 49,5 | 47,1 | 47,0 | 46,7 | 45,9 | 45,6 | 45,5 | 45,5 | 45,2 | 45,3 | 45,4 | 45,3 | 45,3 | 45,3 |
| P.Gap | -0,8 | -0,3 | -0,2 | -0,5 | -0,5 | -0,1 | 0,0 | 0,1 | 0,1 | 0,3 | 0,2 | 0,0 | 0,1 | 0,0 | 0,1 |

The difference in peak temperature for the two scheduling approaches is much smaller than observed on the notebook platform (Table 4.3). While the small temperature range may contribute to this, the behavior appears to differ from the observations on the other platform. The temperatures achieved by the two algorithms doesn't diverge as strongly, even for the highest workload, the GMPT schedules do not appear to provide a strong benefit. As indicated by Figure 4.3, the spread of the data points doesn't differ by a significant amount between the two scheduling approaches. This shows that the frequency changes of the DVFS schedules do not add any instability to the achieved temperature.

On this platform the benefit of the GMPT schedules isn't as pronounced as it appeared on the notebook. In this single-task simulation the peak temperature achieved for low

workloads is, to all intents and purposes, the same for both methods, and even for very high workloads, while there is a difference, it doesn't appear to be very strong.

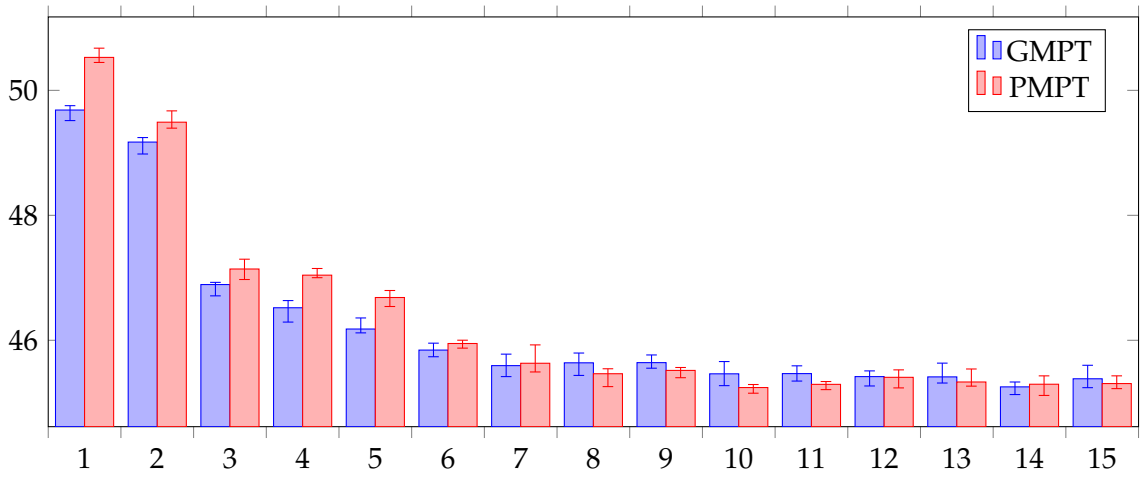



Figure 4.6.: Comparison[3]of the temperature measurements for GMPT and PMPT

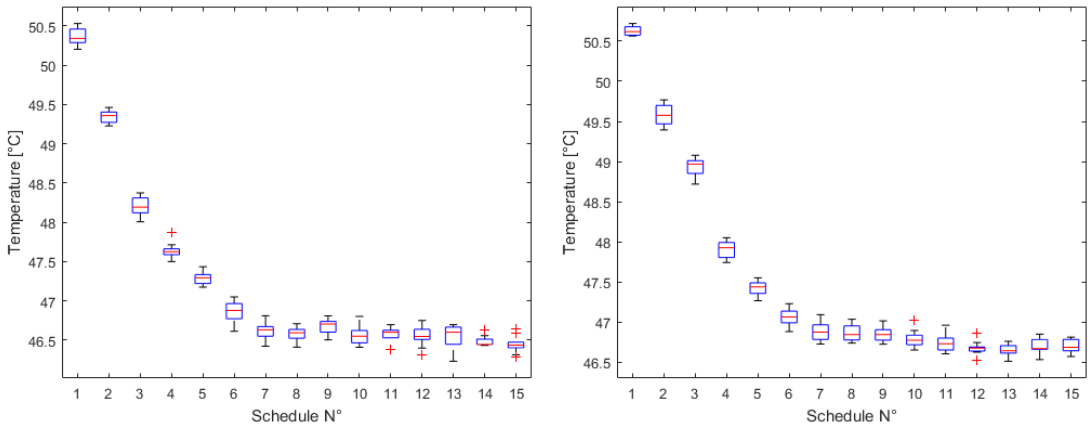
### 4.2.2. Multi-Task

In the multi-task environment on the Raspberry Pi the GMPT schedules are the most complex (Table A.4). They require more frequency changes than those under any of the previous conditions. Based on previous observations, this behavior can partly be traced back to the smaller available frequency and temperature range inherent to the platform in combination with the increased challenge of scheduling multiple tasks.

As illustrated by Figure 4.7 the temperature development over different workloads follows the familiar pattern observed in previous experiments. The peak temperatures achieved by both algorithms is similar throughout all workloads with schedules for low workload achieving practically the same temperature for both approaches. Only the highest workloads yield a noticeable difference for the two scheduling methods, however, like in the single-task simulations, the difference between the two struggles to reach even $1°C$ (Table 4.4).

Similarly to the multi-task simulation on the notebook, the peak temperatures for the PMPT schedules appear to converge toward a slightly higher temperature but unlike on the other platform it is not clear if this difference can be considered significant.

[3]The bar values correspond to the median and the error bars correspond to the lower and upper inner fences displayed in the boxplots in Figure 4.6




(a) Results GMPT (b) Results PMPT

Figure 4.7.: Thermal behavior of schedules in multi-task environment

Table 4.4.: Median peak temperature for all workloads and their performance gap

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| GMPT | 50,3 | 49,4 | 48,2 | 47,6 | 47,3 | 46,9 | 46,6 | 46,6 | 46,7 | 46,5 | 46,6 | 46,5 | 46,6 | 46,4 | 46,4 |
| PMPT | 50,6 | 49,6 | 49,0 | 47,9 | 47,4 | 47,1 | 46,9 | 46,8 | 46,8 | 46,8 | 46,7 | 46,7 | 46,6 | 46,7 | 46,7 |
| P.Gap | -0,3 | -0,2 | -0,8 | -0,3 | -0,1 | -0,2 | -0,3 | -0,2 | -0,1 | -0,3 | -0,1 | -0,2 | 0,0 | -0,3 | -0,3 |

The comparison provided by Figure 4.8 further shows how close the peak temperatures reached by both scheduling approaches are. For most of the lower workloads the error bars indicate that the difference of the medians is not big enough to clearly demonstrate the behavior observed on the notebook. Regarding the higher workloads, the difference in peak temperature is larger and does demonstrate that the schedules derived by GMPT can provide a benefit under some circumstances. Ultimately, whether the slightly lower temperature for the DVFS schedules justifies their application, considering the increased effort associated with their computation, needs to be evaluated on a case-by-case basis.

Before drawing a definite conclusion on this platform, the overhead introduced by the framework needs to be taken into account. The analysis in subsection 3.1.3 has shown, that the frequency changes cause the largest overhead. This mostly affects the DVFS schedules and since it is not clear how highly the CPU is loaded during the execution of the frequency switch, it is not entirely predictable how these lengthy changes affect the temperature development. The results on this platform can therefore not be considered as reliable as the ones from the notebook.

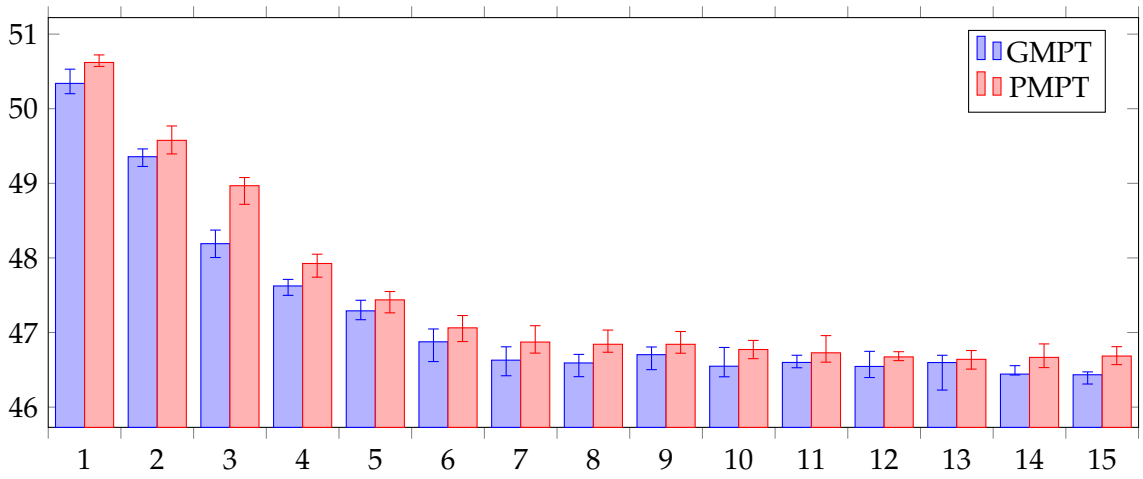



Figure 4.8.: Comparison[4]of the temperature measurements for GMPT and PMPT

## 4.3. Assessment

Both the single-task and the multi-task simulations have shown that the GMPT algorithm can yield schedules that provide a significant benefit in minimizing the CPU peak temperature, especially for high workloads. The DVFS schedules appear to handle the challenges in multi-task environments especially well when compared to the PMPT solutions (based on evidence provided by the notebook experiments). Here these schedules manage to minimize the peak temperature better than their DPM counterparts throughout all workloads.

A further factor that could influence the algorithm choice is the time it takes to compute the schedules. The PMPT algorithm is about one order of magnitude faster than the GMPT algorithm. If the schedules are always computed beforehand, this is not an issue, if, however, the static scheduling scheme needs to be adapted at some point during its execution the PMPT approach may be more advantageous.

The adjustments performed on the notebook and the framework itself succeeded in recreating an environment that resembles a typical hard real-time system. The results provided by the experiments on this platform are therefore considered very reliable and their yields are assumed to be an accurate representation of the performance of both algorithms. The adjustments for the Raspberry Pi on the other hand haven't been as successful. The smaller frequency and temperature range available on this platform

[4]The bar values correspond to the median and the error bars correspond to the lower and upper inner fences displayed in the boxplots in Figure 4.8

makes the comparison of the two approaches undoubtedly more difficult. Additionally, the overhead introduced by the framework is considerably larger than on the notebook, which makes it difficult to draw a clear conclusion regarding the temperature results. Overall the results on this platform cannot be considered as trustworthy as the ones from the notebook. They still provide some evidence that supports the conclusions drawn from the first platform, but when analyzed independently the results are mostly inconclusive.

# 5. Conclusion

This thesis examined a DVFS approach to minimize the processor peak temperature in two different hardware environments.

Initially the framework to be used for future experiments was presented. As part of the setup procedure several changes and adjustments to the framework had to be made. The need for these changes arose in part from the urge to ultimately generate accurate and authentic results which could be reproduced and in part from the necessity of porting the framework to the second hardware platform.

With the framework in working condition on both hardware platforms, the thermal profile, i.e. the behavior of the CPU temperature for different operating frequencies was determined. This data could then be used by a set of algorithms to calculate appropriate schedules for both environments with the aim of minimizing the processor peak temperature.

In the final part of this thesis two algorithms (GMPT and PMPT) were used to calculate aforementioned schedules for different workloads in single-task and multi-task simulations. These schedules were then employed in a series of experiments performed on both hardware platforms. The data gathered in the experiments and specifically the peak temperature achieved for different scheduling schemes was finally used to compare the performance of the GMPT approach to the PMPT method and assess the benefits and drawbacks of the GMPT algorithm.

# 6. Future Work

The experiments performed over the course of this thesis have returned good evidence on the performance of the GMPT and PMPT algorithm. Especially the notebook has provided reliable data which constitutes a good basis upon which the performance of the GMPT algorithm was evaluated. The data from the Raspberry Pi was not as clear.

A first option for future work is the improvement of the framework, specifically its adaption to the Raspberry Pi platform. Currently, experiments on this platform are associated with a major overhead due to the slow frequency switch. This leads to a significant deterioration of the temperature data and makes its clear analysis difficult. As of today, the Raspberry Pi 3B does not provide direct frequency scaling options via the CPUFreq interface (except for setting minimum or maximum frequency) and therefore the framework resorts to the vcmailbox driver functionality, which in its current implementation is the source of the lengthy frequency switch. If the Raspberry Pi is to be used in future experiments for similar purposes the communication with this driver should be looked into, in order to determine potential steps to optimize the interaction with it. Alternatively, future generations of the Raspberry Pi may provide a more flexible CPUFreq driver. In this case, the frequency changes could be implemented in the same fashion as they were on the notebook (likely resulting in similarly good performance).

Ultimately I would propose a different option to work on in the future. The framework may provide an easy way to test these schedules on multiple platforms but it only simulates a real-time system. I don't believe more experiments on the Raspberry Pi will really add a lot of value to the evaluation of the GMPT algorithm and its schedules as Raspberry Pi's themselves are no closer to embedded hard real-time systems than any notebook. To all intents and purposes a Raspberry Pi is just a desktop PC without a fan and without a case. An alternative to this would be to test the schedules on a true real-time systems. Boards, similar to the Raspberry Pi, are available, which run on an actual RTOS. Using such a platform for future experiments requires more setup work but eliminates any influences the OS or the framework have on timing and temperature. These experiments would give unequivocal proof for the performance of the GMPT schedules.

# A. Schedules

Table A.1.: Notebook schedules for different single-task workloads

| Workload[a] | GMPT[b] | PMPT[b] |
|---|---|---|
| WL 1 | ((20, 0.8), (2, 1)) | ((2, 0), (19, 1)) |
| WL 2 | ((4, 0.6), (5, 0.8), (4, 0.6)) | ((7, 0), (19, 1)) |
| WL 3 | ((19, 0.6)) | ((17, 0), (19, 1)) |
| WL 4 | ((12, 0.6), (6, 0.4)) | ((11, 0), (7, 1)) |
| WL 5 | ((2, 0.4), (26, 0.6), (13, 0.4)) | ((25, 0), (13, 1)) |
| WL 6 | ((2, 0.6), (44, 0.4), (38, 0.6)) | ((32, 0), (13, 1)) |
| WL 7 | ((26, 0.4), (10, 0.6), (34, 0.4), (10, 0.6)) | ((37, 0), (13, 1)) |
| WL 8 | ((4, 0.6), (16, 0.4), (16, 0.6), (44, 0.4)) | ((46, 0), (13, 1)) |
| WL 9 | ((18, 0.4), (9, 0.6), (9, 0.4), (1, 0.6), (3, 0.4)) | ((52, 0), (13, 1)) |
| WL 10 | ((31, 0.4), (6, 0.6), (2, 0.8)) | ((33, 0), (7, 1)) |
| WL 11 | ((40, 0.4)) | ((38, 0), (7, 1)) |
| WL 12 | ((4, 0.6), (6, 0.4), (8, 0) (11, 0.4), (10, 0.6)) | ((43, 0), (7, 1)) |
| WL 13 | ((2, 0), (38, 0.4)) | ((48, 0), (7, 1)) |
| WL 14 | ((38, 0.4), (10, 0), (37, 0.4)) | ((53, 0), (7, 1)) |
| WL 15 | ((19, 0.6), (12, 0.4), (16, 0)) | ((58, 0), (7, 1)) |

[a] Workload and the according schedules are sorted from highest to lowest.
[b] Each schedule consists of timeslots represented as tuples in the format (<duration [ms]>, <frequency setting>).

Table A.2.: Notebook schedules for different multi-task workloads

| Workload[a] | GMPT[b] | PMPT[b] |
|---|---|---|
| WL 1 | ((3, 0.8), (15, 1), (12, 0.8)) | ((1, 0), (28, 1)) |
| WL 2 | ((2, 0.8), (4, 0.6), (3, 0.8), (1, 0.6)) | ((7, 0), (22, 1)) |
| WL 3 | ((15, 0.4), (30, 0.6)) | ((7, 0), (11.5, 1)) |
| WL 4 | ((28, 0.4), (2, 0.8), (9, 0.4), (16, 0.6)) | ((13, 0), (16, 1)) |
| WL 5 | ((18, 0.4)) | ((12, 0), (11, 1)) |
| WL 6 | ((7, 0.6), (24, 0.4), (8, 0), (10, 0.6), (1, 0.4)) | ((25, 0), (21, 1)) |
| WL 7 | ((2, 0.6), (7, 0.8), (19, 0), (22, 0.6), (1, 0.4)) | ((25, 0), (21, 1)) |
| WL 8 | ((10, 0), (50, 0.4)) | ((18, 0), (11, 1)) |
| WL 9 | ((7, 0), (2, 0.8), (04, 0.6), (1, 0.4)) | ((22, 0), (13.5, 1)) |
| WL 10 | ((10, 0), (32, 0.4), (1, 0)) | ((18, 0), (9.4, 1)) |
| WL 11 | ((14, 0), (1, 0.8), (30, 0.4)) | ((19, 0), (9.4, 1)) |
| WL 12 | ((26, 0.4), (20, 0), (7, 0.6)) | ((24, 0), (10.5, 1)) |
| WL 13 | ((1, 0.4), (16, 0), (3, 0.4), (07, 0.8)) | ((25, 0), (10.5, 1)) |
| WL 14 | ((1, 0.4), (14, 0.6), (18, 0)) | ((20, 0), (8, 1)) |
| WL 15 | ((08, 0.4), (17, 0), (9, 0.6)) | ((20, 0), (8, 1)) |

[a] Workload and the according schedules are sorted from highest to lowest.
[b] Each schedule consists of timeslots represented as tuples in the format (<duration [ms]>, <frequency setting>).

Table A.3.: Raspberry Pi schedules for different single-task workloads

| Workload[a] | GMPT[b] | PMPT[b] |
|---|---|---|
| WL 1 | ((5, 0.6), (6, 1)) | ((2, 0), (19, 1)) |
| WL 2 | ((1, 1), (20, 0.6), (3, 1), (1, 0.8)) | ((7, 0), (19, 1)) |
| WL 3 | ((10, 0), (30, 0.6)) | ((17, 0), (19, 1)) |
| WL 4 | ((4, 1), (5, 0.4), (9, 0)) | ((11, 0), (7, 1)) |
| WL 5 | ((11, 0), (1, 0.6), (3, 1), (4, 0.6), (1, 0)) | ((25, 0), (13, 1)) |
| WL 6 | ((30, 0), (2, 0.4), (14, 0.8)) | ((32, 0), (13, 1)) |
| WL 7 | ((33, 0), (8, 1), (10, 0.4)) | ((37, 0), (13, 1)) |
| WL 8 | ((13, 0.8), (4, 0.4), (39, 0)) | ((46, 0), (13, 1)) |
| WL 9 | ((28, 0), (1, 0.6), (1, 0.4), (5, 1)) | ((52, 0), (13, 1)) |
| WL 10 | ((4, 0), (7, 1), (29, 0), (7, 1), (33, 0)) | ((33, 0), (7, 1)) |
| WL 11 | ((6, 0), (4, 1), (20, 0)) | ((38, 0), (7, 1)) |
| WL 12 | ((2, 0.8), (42, 0), (3, 1), (1, 0.8), (1, 0.6)) | ((43, 0), (7, 1)) |
| WL 13 | ((1, 0), (10, 0.6), (44, 0)) | ((48, 0), (7, 1)) |
| WL 14 | ((4, 1), (3, 0.8), (51, 0)) | ((53, 0), (7, 1)) |
| WL 15 | ((10, 0), (15, 0.4), (39, 0)) | ((58, 0), (7, 1)) |

[a] Workload and the according schedules are sorted from highest to lowest.
[b] Each schedule consists of timeslots represented as tuples in the format (<duration [ms]>, <frequency setting>).

Table A.4.: Raspberry Pi schedules for different multi-task workloads

| Workload[a] | GMPT[b] | PMPT[b] |
|---|---|---|
| WL 1 | ((20, 1), (1, 0.4), (6, 0.6), (4, 1)) | ((1, 0), (28, 1)) |
| WL 2 | ((5, 1), (1, 0.8), (21, 0.6), (2, 1)) | ((7, 0), (22, 1)) |
| WL 3 | ((15, 0.4), (30, 0.6)) | ((7, 0), (11.5)) |
| WL 4 | ((44, 0.6), (11, 0)) | ((13, 0), (16, 1)) |
| WL 5 | ((31, 0.4)) | ((12, 0), (11, 1)) |
| WL 6 | ((1, 0.6), (23, 0.8), (24, 0), (1, 1)) | ((25, 0), (21, 1)) |
| WL 7 | ((21, 0), (2, 0.8), (18, 1), (1, 0.4), (5, 0)) | ((25, 0), (21, 1)) |
| WL 8 | ((1, 0), (2, 0.6), (9, 1), (17, 0)) | ((18, 0), (11, 1)) |
| WL 9 | ((20, 0), (6, 0.8), (6, 1), (1, 0.8)) | ((22, 0), (13.5)) |
| WL 10 | ((11, 0), (1, 1), (12, 0.4), (7, 1), (10, 0)) | ((18, 0), (9.4)) |
| WL 11 | ((1, 0.4), (11, 1), (1, 0.8), (1, 0.4), (23, 0)) | ((19, 0), (9.4)) |
| WL 12 | ((1, 0.4), (4, 0.8), (23, 0), (6, 1)) | ((24, 0), (10.5)) |
| WL 13 | ((1, 0), (9, 0.8), (17, 0)) | ((25, 0), (10.5)) |
| WL 14 | ((4, 1), (23, 0), (1, 0.4), (4, 1), (2, 0.6)) | ((20, 0), (8, 1)) |
| WL 15 | ((23, 0), (4, 1), (1, 0.4), (4, 1)) | ((20, 0), (8, 1)) |

[a] Workload and the according schedules are sorted from highest to lowest.
[b] Each schedule consists of timeslots represented as tuples in the format (<duration [ms]>, <frequency setting>).

# List of Figures


2.1. DPM Schedules . . . . . 5
2.2. DPM Temperature Development . . . . . 5
2.3. DVFS Schedules . . . . . 8
2.4. GMPT Schedule . . . . . 11
2.5. GMPT Temperature Development . . . . . 11

3.1. Architecture . . . . . 14
3.2. Framework Schedule . . . . . 15
3.3. Command Time . . . . . 16
3.4. Function Fit . . . . . 19
3.5. Frequency Ranges . . . . . 20
3.6. Thermal Profile Notebook . . . . . 23
3.7. Thermal Profile Raspberry Pi 3B . . . . . 24

4.1. Results Single-Task Notebook . . . . . 26
4.2. Comparison GMPT PMPT . . . . . 27
4.3. Results Multi-Task Notebook . . . . . 28
4.4. Comparison Multi-Task Notebook . . . . . 29
4.5. Results Single-Task Raspberry . . . . . 30
4.6. Comparison Single-Task Raspberry . . . . . 31
4.7. Results Multi-Task Raspberry . . . . . 32
4.8. Comparison Multi-Task Raspberry . . . . . 33

# List of Tables

3.1. Overhead . . . . . 18
3.2. Thermal Profile Notebook . . . . . 22
3.3. Thermal Profile Raspberry Pi . . . . . 23

4.1. Single-Task median results Notebook . . . . . 26
4.2. Multi-Task Notebook median results . . . . . 28
4.3. Single-Task median results Raspberry . . . . . 30
4.4. Multi-Task median results Raspberry . . . . . 32

A.1. Single-Task Notebook schedules . . . . . 39
A.2. Multi-Task Notebook schedules . . . . . 40
A.3. Single-Task Pi schedules . . . . . 41
A.4. Single-Task Pi schedules . . . . . 42

# Bibliography


[1] M. Ahmed, N. Fisher, S. Wang, and P. Hettiarachchi. "Minimizing peak temperature in embedded real-time systems via thermal-aware periodic resources." In: *Sustainable Computing: Informatics and Systems* (2011), pp. 226–240.

[2] S. K. Baruah, A. K. Mok, and L. E. Rosier. "Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor." In: *Real-Time Systems Symposium. 11th Proceedings* (1990), pp. 182–190.

[3] L. Bersentes. "Evaluation of a realtime system towards SF3P / Litmus RT and Real-Time Temperature Analysis." PhD thesis. Technische Universität München, 2017.

[4] D. Brodowski, N. Golde, R. J. Wysocki, and V. Kumar. *Linux CPUFreq, https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt*.

[5] D. Brooks and M. Martonosi. "Dynamic thermal management for high-performance microprocessors." In: *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*. 2001, pp. 171–182.

[6] T. Chantem, X. S. Hu, and R. P. Dick. "Online work maximization under a peak temperature constraint." In: *Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design*. 2009, p. 105.

[7] V. Chaturvedi, H. Huang, and G. Quan. "Leakage Aware Scheduling on Maximum Temperature Minimization for Periodic Hard Real-Time Systems." In: *10th IEEE International Conference on Computer and Information Technology*. 2010, pp. 1802–1809.

[8] J.-J. Chen, S. Wang, and L. Thiele. "Proactive Speed Scheduling for Real-Time Tasks under Thermal Constraints." In: *15th IEEE Real-Time and Embedded Technology and Applications Symposium*. 2009, pp. 141–150.

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

[10] S. Cheng, J. A. Stankovic, and K. Ramamritham. *Scheduling Algorithms for Hard Real-Time Systems–A Brief Survey*. 1987.

[11] G. Dhiman and T. S. Rosing. "Dynamic voltage frequency scaling for multi-tasking systems using online learning." In: *Proceedings of the international symposium on Low power electronics and design*. 2007, pp. 207–212.

[12] *Enhanced Intel ® SpeedStep ® Technology for the Intel ® Pentium ® M Processor, http://download.intel.com/design/network/papers/30117401.pdf*. 2004.

[13] J. H. Holland. "Genetic Algorithms." In: *SCIENTIFIC AMERICAN* (1992).

[14] M. Kumar, M. Husian, N. Upreti, and D. Gupta. "GENETIC ALGORITHM: REVIEW AND APPLICATION." In: *International Journal of Information Technology and Knowledge Management* (2010), pp. 451–454.

[15] P. Kumar and L. Thiele. "Thermally optimal stop-go scheduling of task graphs with real-time constraints." In: *Proceedings of the Asia and South Pacific Design Automation Conference*. 2011, pp. 123–128.

[16] E. Le Sueur and G. Heiser. *Dynamic voltage and frequency scaling: the laws of diminishing returns*. 2010.

[17] C. Mack. "The Multiple Lives of Moore's Law." In: *IEEE Spectrum* (2015), pp. 31–31.

[18] *Mailbox property interface https://github.com/raspberrypi/firmware/wiki/Mailbox-property-interface*. 2017.

[19] D. Meisner, B. T. Gold, and T. F. Wenisch. "PowerNap: Eliminating Server Idle Power." In: *ASPLOS XIV Proceedings of the 14th international conference on Architectural support for programming languages and operating systems*. 2009, pp. 205–216.

[20] M. Mohaqeqi, M. Kargahi, and A. Movaghar. "Analytical Leakage-Aware Thermal Modeling of a Real-Time System." In: *IEEE Transactions on Computers* (2014), pp. 1378–1392.

[21] G. E. Moore. "Cramming More Components onto Integrated Circuits." In: *Electronics* (1965), pp. 114–117.

[22] G. E. Moore. "Progress in digital integrated electronics." In: *Electron Devices Meeting, International*. 1975, pp. 11–13.

[23] A. Mutapcic, S. Boyd, S. Murali, D. Atienza, G. De Micheli, and R. Gupta. "Processor Speed Control With Thermal Constraints." In: *IEEE Transactions on Circuits and Systems I: Regular Papers* (2009), pp. 1994–2008.

[24] V. Pallipadi and A. Starikovskiy. "The Ondemand Governor Past, Present, and Future." In: *Proceedings of Linux Symposium*. 2006, pp. 223–238.

[25] K. Pratyush and L. Thiele. “Cool shapers: Shaping real-time tasks for improved thermal guarantees.” In: *Design Automation Conference (DAC)*. 2011, pp. 468–473.

[26] Robingroppe. *GitHub raspberrypi/linux RPi3 Frequency Scaling issue #1335 https://github.com/raspberrypi/linux/issues/1335*. 2016.

[27] G. Semeraro, G. Magklis, R. Balasubramonian, D. Albonesi, S. Dwarkadas, and M. Scott. “Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling.” In: *Proceedings Eighth International Symposium on High Performance Computer Architecture*, pp. 29–40.

[28] K. Skadron, M. Stan, W. Huang, Sivakumar Velusamy, Karthik Sankaranarayanan, and D. Tarjan. “Temperature-aware microarchitecture.” In: *30th Annual International Symposium on Computer Architecture. Proceedings.* 2003, pp. 2–13.

[29] J. Srinivasan, S. Adve, P. Bose, and J. Rivers. “The impact of technology scaling on lifetime reliability.” In: *International Conference on Dependable Systems and Networks*. 2004, pp. 177–186.

[30] Sushu Zhang and K. S. Chatha. “Approximation algorithm for the temperature-aware scheduling problem.” In: *IEEE/ACM International Conference on Computer-Aided Design*. 2007, pp. 281–288.

[31] T. N. Theis and H.-S. P. Wong. “The End of Moore’s Law: A New Beginning for Information Technology.” In: *Computing in Science & Engineering* (2017), pp. 41–50.

[32] *VideoCore Tools, https://github.com/nezticle/RaspberryPi-BuildRoot/wiki/VideoCore-Tools*. 2014.

[33] Y. Xiang, T. Chantem, R. P. Dick, X. S. Hu, and L. Shang. “System-level reliability modeling for MPSoCs.” In: *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. 2010, p. 297.

[34] M. Zhou. *Minimizing Peak Temperature for Hard Real-time Systems Based on Combinatorial Optimization*. 2017.