MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Hornof David** Personal ID number: **466001**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Software Engineering**

II. Master's thesis details

Master's thesis title in English:

Offline scheduling of the safety-critical tasks within the isolation time-windows

Master's thesis title in Czech:

Offline rozvrhování kritických úloh v rámci časových oken zajišťujících jejich izolovanost

Guidelines:

With the growth of the automotive and avionics industries, demands on the embedded platforms are increasing. The developers need to address various reliability and safety issues to meet strict standards. This diploma thesis addresses a scheduling problem where the safety-critical tasks need to be scheduled within the isolation time-windows on a heterogeneous embedded platform (i.e., a platform having multiple distinct computational clusters (CPU, GPU) of various characteristics). The schedule is created offline, assuming the worst-case execution times of the tasks, which allows proper validation before the real deployment. Creating a schedule by hand might often be hard, especially when many tasks are considered. Besides the automation, the goal of this thesis is, for a given instance, to find a schedule that would minimize the steady-state temperature of the platform while being repeatedly executed. This can be achieved by properly allocating and scheduling the tasks on the platform's computational clusters [1,2].
The student will:
1. review the existing scientific literature addressing the topics of static (offline) scheduling with isolation windows and energy-aware task allocation;
2. propose a formal (mathematical) model of the problem (e.g., integer linear programming) optimizing the steady-state temperature of the schedule;
3. design, implement and test the necessary software tools allowing the user to generate problem instances, run algorithms, evaluate and visualize the results, etc.;
4. implement the model, and empirically validate its performance on a real physical platform (e.g., i.MX8 by NXP [4]);
5. create a set of benchmark instances, and compare the performance of the proposed model with other models or rule-based algorithms, which are typically used in the literature.

Bibliography / sources:

1. K. Chin-Fu, L. Yung-Feng, "Task assignment with energy efficiency considerations for non-DVS heterogeneous multiprocessor systems," 2015 SIGAPP Appl. Comput. Rev. 14, 4, pp. 8–18.
2. Sumarga Kumar Sah Tyagi, Deepak Kumar Jain, Steven Lawrence Fernandes, Pranab K. Muhuri, "Thermal-aware power-efficient deadline based task allocation in multi-core processor," 2017, Journal of Computational Science, 19, pp. 112 - 120.
3. P. Huang, G. Giannopoulou, R. Ahmed, D. B. Bartolini and L. Thiele, "An Isolation Scheduling Model for Multicores," 2015 IEEE Real-Time Systems Symposium, San Antonio, TX, 2015, pp. 141-152.
4. NXP.. i.MX 8QuadMax/QuadPlus Multisensory Enablement Kit, [Online]. Available:https://www.nxp.com/design/development-boards/i-mx-evaluation-and-development-boards/i-mx-8quadmax-multisensory-enablement-kit-mek:MCIMX8QM-CPU.

Name and workplace of master's thesis supervisor:

Ing. Ondřej Benedikt, Department of Control Engineering, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **10.02.2021** Deadline for master's thesis submission: **21.05.2021**

Assignment valid until: **30.09.2022**

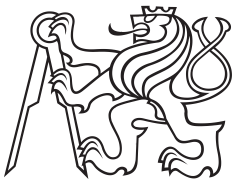
Ing. Ondřej Benedikt Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Offline scheduling of the safety-critical tasks within the isolation time-windows

Bc. David Hornof

Supervisor: Ing. Ondřej Benedikt
Field of study: Open Informatics
Subfield: Software Engineering
May 2021

Acknowledgements

First and foremost, I would like to sincerely thank my supervisor Ondřej Benedikt for all his time, guidance, and advice he has given me.

I would also like to thank my family for providing me with an environment that has allowed me to focus on my studies.

Last but not least, I would like to thank my friend Jan Kuběna for his help with proofreading.

Declaration

I hereby declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 21, 2021

Abstract

The requirements put on embedded systems are ever-increasing. As these systems are being tasked with handling an increasing number of safety-critical tasks, their reliability must be guaranteed. It is therefore vital to prevent them from overheating. In this thesis, we explore thermal-aware scheduling of periodic safety-critical workloads to isolation windows. We propose an empirical model for estimating power consumption of an MPSoC based system that allows us to formulate the scheduling problem as an ILP optimization problem for minimizing the system's power consumption, which leads to minimization of the system's steady-state temperature. We design and implement software to help us prepare experiments to evaluate our method. Subsequently, we perform the evaluation of the method on a hardware platform powered by the i.MX8 processor supplied by NXP. The results show that schedules produced by our method can result in steady-state temperatures lower by up to 12% when compared to other tested methods and around 3% lower compared to a method adapted from literature. We also show how changing a schedule length can affect the steady-state temperature of the system.

Keywords: offline scheduling, thermal-aware scheduling, thermal-aware task mapping, MPSoC, safety-critical workloads, isolation windows, ARINC 653

Supervisor: Ing. Ondřej Benedikt

Abstrakt

Požadavky kladené na vestavěné systémy se stále zvyšují. Vzhledem k tomu, že tyto systémy obsluhují rostoucí množství úloh kritických pro bezpečnost, jejich spolehlivost musí být zaručena a nesmí proto docházet k jejich přehřívání. V této práci se zabýváme rozvrhováním periodických úloh kritických pro bezpečnost do izolačních oken s ohledem na ustálenou teplotu systému. Navrhli jsme empirický model pro odhadování příkonu systému s heterogenní architekturou, který jsme využili k zformulování naší rozvrhovací úlohy jakožto ILP optimalizačního problému pro minimalizaci příkonu systému. Minimalizací příkonu je možné dosáhnout minimalizace ustálené teploty systému. Dále jsme navrhli a implementovali software, který nám usnadnil přípravu experimentů pro vyhodnocení kvality naší metody. Kvalitu této metody jsme následně vyhodnotili na hardwarové platformě s procesorem i.MX8 od NXP. Výsledky ukázaly, že rozvrhy vytvořené naší metodou dosáhly ustálených teplot nižších až o 12% v porování s jinými testovanými metodami a okolo 3% nižších než existující rozvrhovací metoda adaptovaná z literatury. Dále jsme v práci ukázali, jak délka rozvrhu může ovlivnit ustálenou teplotu systému.

Klíčová slova: offline rozvrhování, rozvrhování zohledňující teploty, mapování úloh zohledňující teploty, MPSoC, bezpečnostně kritické úlohy, izolační okna, ARINC 653

Překlad názvu: Offline rozvrhování kritických úloh v rámci časových oken zajišťujících jejich izolovanost

Contents

Figures

Tables

Chapter 1

Introduction

As the automotive and avionics industries grow with the technological advances of recent years, so do the demands on embedded systems. Nowadays, these systems are being tasked with handling more and more safety-critical workloads whilst often operating under harsh environmental conditions. As the workload of these systems increases, so must their performance in order to satisfy the increasing requirements. However, with the performance increase the produced waste heat increases as well. Since reliability is one of the key requirements for systems handling safety-critical workloads, to satisfy it, the systems must operate within a given thermal envelope. Active cooling is one of the most straightforward ways to guarantee the thermal envelope. However, implementing it increases the mechanical complexity which then results in increased costs. As such, it is beneficial to explore passive cooling as an alternative approach.

An emerging trend in the hardware platforms used for embedded systems is the use of so-called Multiprocessor System on a Chip (MPSoC). As the name suggests, these are chips containing multiple different microprocessor cores. Usually, the chips consist of cores that share the same instruction set, which means that the same workload can easily be executed on any of the cores. One of the possible MPSoC setups is a combination of cores offering high performance at the cost of high energy demand (resulting in increased heat produced), with cores that are more energy efficient in exchange for lower computational performance. On heterogeneous platforms utilizing MPSoCs, a clever allocation of the workload to the available processor cores can be used as a passive cooling technique to reduce the system's temperature.

In this thesis, we explore thermal-aware scheduling of safety-critical workloads that are executed periodically. Our main goal is to design, implement and evaluate a method for creating a schedule in an offline manner, which minimizes the steady-state temperature of the system. We are interested in the behavior of real hardware, and thus we rely mostly on experiments and empirical observations to achieve this goal.

In Chapter 2, we introduce the hardware we use for experiments, explain our scheduling problem and how we can minimize the steady-state temperature of the system by minimizing its average power consumption. Next in Chapter 3, we explore the existing research in the field that is relevant to the problem we

are solving. In Chapter 4 we formalize our problem and introduce the PEWMS model, which we use to estimate the power consumption of our system. To find a schedule minimizing the system's average power consumption with the use of the PEWMS model, we formulate the task as an optimization problem with ILP formalism in Chapter 5. In this chapter, we also introduce alternative methods for creating a schedule, which we use for comparison with the method based on the PEWMS model.

In Chapter 6, we describe the software tools we created to help us prepare experiments for the evaluation of our model. We then perform the evaluation in Chapter 7. We look at the thermal performance of the schedules we prepared and at the power consumption estimation accuracy of the PEWMS model. We also perform experiments demonstrating a tradeoff between schedule length and steady-state temperature of the system in Chapter 8. We conclude this thesis by summarizing our results and discussing the future work in Chapter 9.

The main contributions of this work are:

- A proposal of a thermal-aware scheduling method for periodic safety-critical workloads.
- An introduction of a model for estimating power consumption based on empirical observations, which serves as a base for our thermal-aware scheduling method.
- Experimental evaluation of both the scheduling method and the power-estimation model performed on real hardware.
- Assessment of a tradeoff between schedule length and temperature.

Chapter 2
Preliminaries

We begin by introducing the hardware platform we use for experimenting in Section 2.1. Then we present our scheduling model and related terminology in Section 2.2 and explain the ideas behind thermal-aware scheduling in Section 2.3 and Section 2.4.

2.1 Hardware platform

The core of our testbed is the i.MX8 QuadMax Multisensory Enablement Kit (MEK) board supplied by NXP [1]. It is based around the i.MX8 QuadMax multiprocessor system on a chip (MPSoC), which is the latest generation of the i.MX MPSoC family from NXP. We chose this particular system as it is used in numerous areas, including the automotive and avionics domains.

The i.MX8 QuadMax is a representative of ARM's big.LITTLE architecture and offers two computing clusters. The first cluster consists of two ARM Cortex-A72 cores, while the second cluster consists of four ARM Cortex-A53 cores. The i.MX8 QuadMax also offers two GPUs. However, we only address thermal-aware scheduling of tasks that do not use them in this thesis. Figure 2.1 shows a block diagram of the computing cluster arrangement in the i.MX8 QuadMax chip.

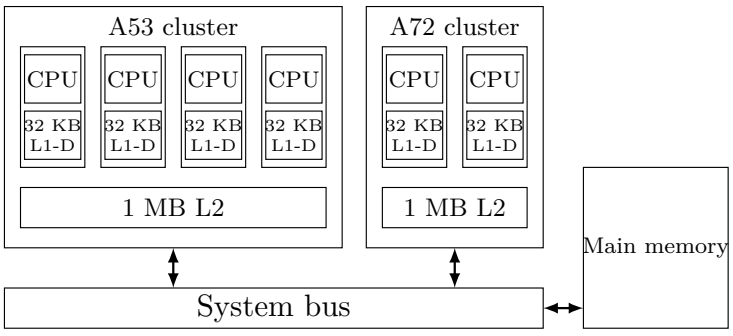


Figure 2.1: Block diagram of the i.MX8 QuadMax computing cluster arrangement

Our testbed is also equipped with an ambient temperature sensor and an external power meter module, which we use to measure the power consumption

of the MEK board. The power meter module uses the INA219 chip [2] to measure the power consumption. We use built-in sensors of the i.MX8 chip to measure the chip's temperature. The testbed setup is shown in Figure 2.2 – MEK is the largest board in the picture.

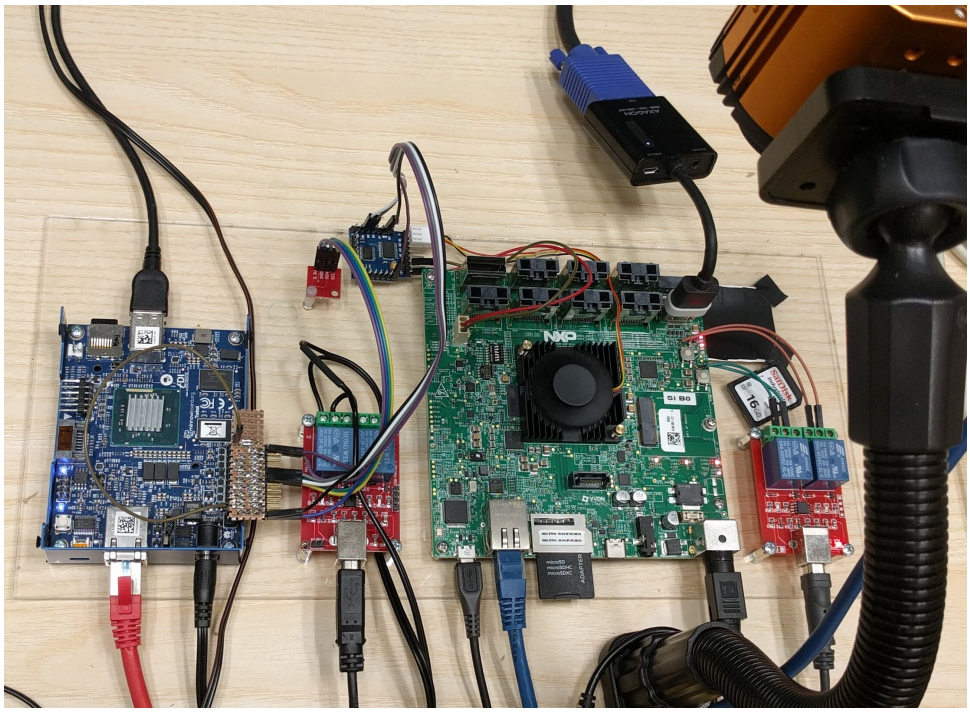


Figure 2.2: Testbed setup

2.2 Scheduling model

We are tasked with creating a schedule for a set of *safety-critical partitions*. A safety-critical partition is a set of processes that must be executed periodically. Otherwise, the safety of the system would be compromised. The safety-critical partitions are to be scheduled within *isolation windows*. An isolation window spans across all CPU cores of the system. Inside of an isolation window, at most one safety-critical partition can be executed on a single CPU core. The set of these isolation windows is known as a *major frame*. We call the mapping of the safety-critical partitions to isolation windows and CPU cores, along with the fixed order in which the windows are executed a *schedule*. We assume that all safety-critical partitions are ready at the start and must be completed before a common deadline denoted h, known as the *major frame length*. The major frame is repeatedly executed on the system. Figure 2.3 shows a valid major frame schedule of six safety-critical partitions (T_1-T_6) respecting the described rules.

The introduced model is motivated by the ARINC 653 standard [3]. This standard is used in the avionics domain and deals, among others, with

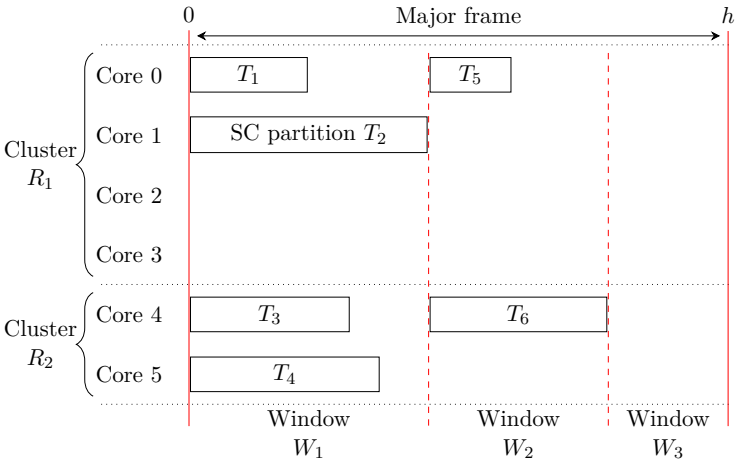


Figure 2.3: Valid schedule of a major frame

space and time partitioning in real-time operating systems. It specifies that processes on the system are grouped into partitions, which are then scheduled into windows. The standard also specifies implementation details, such as the behavior of an operating system scheduler or handling of interprocess communication.

2.3 Thermal-aware scheduling

The idea of thermal-aware scheduling is not new. Sheikh et al. [4] provide an overview of achieved results and explain the common approaches to modeling the thermal behavior of a system.

We can assume that our system will spend enough time executing the major frame to reach its steady-state temperature since the major frame lengths we work with are significantly shorter than the system's operating time. According to [4], under this assumption, the steady-state temperature of the system can be estimated as:

$$T_\infty = T_{amb} + C \cdot P \tag{2.1}$$

where T_∞ is the steady-state temperature, C is a constant dependent on the hardware specification, P is the power consumption, and T_{amb} is the ambient temperature. Note that this is a simplified model that considers the entire system to be a single thermal node and is not entirely accurate. Nevertheless, it is clear from Equation (2.1) that power consumption and steady-state temperature are directly connected. Thus, we can minimize the steady-state temperature of the system by minimizing its power consumption.

2.4 Power consumption minimization

The most common methods for reducing power consumption of a CPU include dynamic voltage and frequency scaling (DVFS) and dynamic power

management (DPM), as stated by Bambagini et al. [5]. DVFS allows for a trade-off between computational performance and power consumption by altering the CPU frequency and supply voltage. The idea of DPM is to switch the CPU to a low-power state when it is inactive. Another way of reducing power consumption available on heterogeneous systems is mapping tasks to different computing clusters. Depending on the system's architecture, a trade-off between power consumption and execution time of a task can be achieved.

Due to the requirements of our industrial partner, all computing clusters of our CPU run on a fixed frequency that is decided in advance. Use of DVFS is forbidden as it would compromise predictability of the workload execution.

We will not consider the effects of DPM either due to the limitations of our software as it does not allow us to directly control the power state of the CPU. Therefore, we are going to focus on controlling the system's power consumption only by mapping the safety-critical partitions to the different computing clusters our MPSoC offers.

Chapter 3

Related work

As we pointed out in Section 2.3, research in the area of thermal-aware scheduling already exists. Likewise, methods for minimizing average power consumption of a system and the related problem of energy-aware scheduling have also been explored. Most of the research is centered around real-time systems. In this chapter we discuss the works we consider to be relevant to our cause.

3.1 Thermal and energy-aware scheduling

A common approach to solving the problem of thermal and energy-aware scheduling is to present a thermal or a power model of the system. Based on these models, the authors usually develop a mathematical model of the system which can be used for solving the relevant optimization problem.

For example, Chen et al. [6] propose a system model which they develop into an integer linear programming (ILP) model for energy minimization. They note that the ILP model can suffer from performance issues due to its complexity and present a simplified model and several heuristic algorithms for approximating solutions of their ILP model. However, the authors focus on the quality of solution of their heuristic algorithms when compared to their ILP model and do not test their energy minimization model on real hardware. They also do not specify how to obtain some parameters of the model, such as the power characteristic of a task. Similarly, Qin et al. [7] present a generalized system model, which they also develop into an ILP model. Their model relies heavily on the DVFS capabilities of the target system. They also note the performance issues a complex ILP model suffers from.

Another approach is to create an algorithm directly solving the assignment or the scheduling problem studied. Zhou et al. [8] present a greedy heuristic algorithm, which works by assigning tasks to processors based on their expected energy consumption. While scheduling, they insert idle periods into the schedule as a way of lowering the peak temperature of the system. Kuo et al. [9] propose a greedy heuristic algorithm which is also based on the idea of processing tasks in order of their expected energy consumption. In this algorithm, the tasks are processed in order from the most energy

demanding to the least demanding one. For each task, the best (the least energy-demanding) assignment to a processor possible is selected.

A broader overview of results in the areas of thermal and energy-aware scheduling can be found in [4, 5, 10]. As for the thermal and energy models, the main ideas are not too different across the referenced works. The differences lie mostly in the complexity of adopted models. A common issue among the papers is not specifying a way of obtaining some of the parameters of the models. A prominent example of this issue is a coefficient representing the energy consumption behavior of a task used in [6, 8, 9]. Neither of these papers discusses how this parameter is obtained despite its importance.

3.2 Time-partitioned scheduling

To the best of our knowledge, a scheduling model similar to ours (as described in Section 2.2) has not been studied in the context of thermal or energy optimization on heterogeneous systems. This is not true for the general idea of time-partitioned scheduling. For example, Huang et al. [11] group tasks into different categories. Tasks from each category are then executed in isolation windows together. Their motivation is optimizing the usage of shared resources of a real-time system, such as memory cache. While these issues may be relevant for us, the method of creating isolation windows presented does not help us with the problem at hand because it deals with neither temperatures nor power consumption of the system.

The scheduling model we use loosely resembles the scheduling of so-called batching machines, which are typically used in semiconductor manufacturing. We can think of fitting tasks to isolation windows as creating batches on a batching machine – a window would correspond to a batch on the batching machine. Batching machine schedule optimization problems are usually not concerned with optimizing temperatures or power consumption. Instead, more traditional goals for scheduling problems are pursued, such as minimizing makespan or the maximum lateness of the schedule. These problems have been widely studied, and a comprehensive summary of results was published by Brucker et al. [12].

Chapter 4

Problem formalization

In this chapter, we formalize the presented problem. We start by introducing notation in Section 4.1. Then we describe the Power Estimation With Max Static power term (PEWMS) model, which we created based on empirical observations in Section 4.2 and finally, we provide the formal problem statement in Section 4.3.

4.1 Notation

Let $\mathcal{R} = \{R_1, R_2, ..., R_m\}$ be a set of computational resources, where each resource has $c_k \in \mathbb{N}$ processing units. Resource R_k corresponds to a computing cluster of the real system and its capacity to the number of the cluster's cores. We consider the processing units of a computational resource to be identical.

Let $\mathcal{T} = \{T_1, T_2, ..., T_n\}$ be a set of safety-critical partitions to be scheduled. For the sake of simplicity, we will henceforth refer to them simply as *tasks* and assume that each partition consists only of a single process. The execution time of task T_i when executed on resource R_k is $p_{i,k} \in \mathbb{N}$. Tasks are non-preemptive and cannot be terminated early.

Let $\mathcal{W} = \{W_1, W_2, ..., W_\ell\}$ be the set of isolation windows that constitute the major frame. Let $l_j \in \mathbb{N}$ denote the length of isolation window W_j. Let $h \in \mathbb{N}$ be the major frame length. The following holds:

$$\sum_{W_j \in \mathcal{W}} l_j = h \tag{4.1}$$

It is possible that no tasks are assigned to a window. We call such window *empty*. We allow the set $\mathcal{W}$ to contain at most one empty window, since according to the PEWMS power model we introduce in Section 4.2, multiple empty windows can be merged together without affecting the average power consumption of the major frame. Note that the value ℓ (the number of isolation windows) is not known in advance. Despite that, we can define its upper-bound as:

$$\ell \leq |\mathcal{T}| + 1 \tag{4.2}$$

Since in the worst case, each window $W_j \in \mathcal{W}$ will contain only a single task. The addition of one accounts for the possibility of an empty window.

Each task T_i is assigned to be executed on resource R_k in isolation window W_j. Each task is assigned to a single resource (always requiring one processing unit) and to a single window. We use functions $a_r : \mathcal{T} \to \mathcal{R}$ and $a_w : \mathcal{T} \to \mathcal{W}$ to model these assignments. These assignment functions are not known in advance. The length of window W_j must be no less than the execution time of the longest task assigned to it:

$$l_j \geq \max_{\substack{T_i \in \mathcal{T} \\ R_k \in \mathcal{R}}} \{p_{i,k} | a_w(T_i) = W_j \wedge a_r(T_i) = R_k\} \tag{4.3}$$

The *schedule* is given by the assignment functions a_r and a_w and the fixed order in which the isolation windows are to be executed.

4.2 The PEWMS power model

As discussed in Section 3.1, a common issue we encountered when exploring existing research is the lack of discussion with regard to obtaining the parameters relating to the energy consumption behavior of a task. We are going to mitigate this by proposing a simple power model based on empirical observations.

Our model is based on the model proposed by Chen et al. [6] which considers power consumption to be composed of two parts: static power consumption and dynamic power consumption. According to the authors, static power consumption is a constant that represents the power needed for the system to maintain its activity, while dynamic power consumption is directly proportional to the system's activity.

We will adopt a similar idea of static and dynamic power consumption for our model.[1] Unlike [6], we presume that the static power consumption is influenced by the system's activity as well. We do not model DVFS or DPM due to the reasons stated in Section 2.4.

When the system is completely idle, it consumes power P_{idle}. This is the base static power consumption of the system, which can be measured.

We characterize the power consumption behavior of task T_i on resource R_k by two coefficients: $a_{i,k}$ and $b_{i,k}$. Coefficient $a_{i,k}$ corresponds to the dynamic power consumption characteristic of the task, while coefficient $b_{i,k}$ represents the increase in static power consumption of the system when executing task T_i. We obtain these values from measurements for each resource $R_k \in \mathcal{R}$ and task $T_i \in \mathcal{T}$: the task in question is executed first on a single processing unit of resource R_k and then on c_k processing units of this resource. Power consumption is measured each time and a linear function $P_{i,k}(x) = a'x + b'$ is fitted through the measured values, where $P_{i,k}(x)$ is the measured power consumption and x is the number of processing units used. We identify the

[1] The terms static and dynamic power are usually used in the context of modeling behavior of CMOS devices where these terms correspond to characteristics rooted in the physical properties of the modeled device. This is not the case in our model – we use them in a looser way. Furthermore, we do not consider *leakage power*, which is another component used when modeling CMOS devices.

slope of the function a' with dynamic power consumption characteristic of the task and the intercept of the function b' with the static power consumption of the system when the task was being executed. The coefficient $a_{i,k}$ is therefore equal to a'. Since we are interested only in the characteristics of the task itself, we correct the b' value by subtracting the base static power consumption P_{idle} to obtain coefficient $b_{i,k}$: $b_{i,k} = b' - P_{idle}$. This process is illustrated in Figure 4.1.

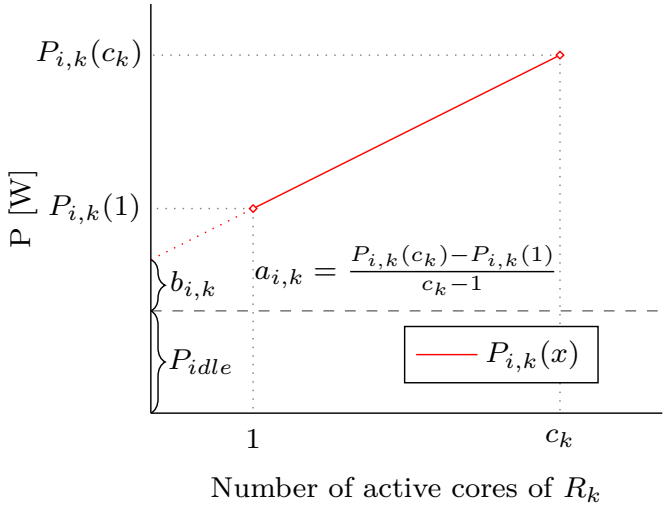


Figure 4.1: Interpolation of the power consumption coefficients

Given these values, we estimate the power consumption P of the system while running a single task T_i on resource R_k as:

$$P = P_{idle} + a_{i,k} + b_{i,k} \tag{4.4}$$

Next, we need to estimate the power consumption of running multiple tasks, e.g., T_i and T_j, on the system. Based on empirical observations, we have determined that the dynamic power coefficients add up, while only the maximum increase in static power is effective.[2] Therefore, the estimated power consumption of the system running tasks T_i and T_j is:

$$P = P_{idle} + a_{i,k} + a_{j,k} + \max\{b_{i,k}, b_{j,k}\} \tag{4.5}$$

The next step is extending the estimation to cover the entire major frame. Until now, our estimations assumed that the tasks were running indefinitely. This is not the case in the major frame, as during the course of different isolation windows, different tasks are executed on the computational resources. To overcome this, we will compute the estimated energy consumption of each window, which we can use to compute the average power consumption of the major frame.

[2]Comparison with alternative power estimation functions that we experimented with can be found in Appendix C.

To estimate the energy consumption of an isolation window, we need to know its dynamic power consumption characteristic and its effect on the static power consumption of the system. We use Equation (4.5) to obtain these values as the same assumptions apply. Each task T_i in window W_j is active for $p_{i,k}$ time units. We assume that the increase in static power consumption of the system is effective during the course of the entire window. While this may lead to overestimating the energy consumption of window W_j, it keeps the estimation function relatively simple:

$$E_j = l_j \cdot P_{idle} + \sum_{T_i \in \mathcal{T}: a_w(T_i) = W_j} \sum_{R_k \in \mathcal{R}: a_r(T_i) = R_k} p_{i,k} \cdot a_{i,k} + l_j \cdot \max_{\substack{T_i \in \mathcal{T}: a_w(T_i) = W_j \\ R_k \in \mathcal{R}: a_r(T_i) = R_k}} b_{i,k} \tag{4.6}$$

We can now estimate the average power consumption of the entire major frame as:

$$P_{MF} = \frac{1}{h} \sum_{W_j \in \mathcal{W}} E_j \tag{4.7}$$

We will refer to the described model as PEWMS, short for Power Estimation With Max Static power term.

4.3 Problem statement

The goal is to find a schedule (which consists of task mapping functions a_r and a_w and the execution order of the isolation windows) that minimizes the average power consumption of the major frame according to the PEWMS power model (Equation (4.7)). In the PEWMS model, the order of windows is arbitrary, as the average power consumption of the major frame depends only on the task mapping. Therefore the actual goal is to find a feasible task mapping, such that the average power consumption of the major frame is minimized. The input consists of the sets $\mathcal{R}$, $\mathcal{T}$, and the value h.

We consider the task mapping to be feasible if and only if the following conditions are satisfied:

- **Task assignment:** Each task $T_i \in \mathcal{T}$ is assigned to exactly one resource $R_k \in \mathcal{R}$ and exactly one window $W_j \in \mathcal{W}$.

$$\forall T_i \in \mathcal{T} \;\; \exists! R_k \in \mathcal{R} : a_r(T_i) = R_k \tag{4.8}$$

$$\forall T_i \in \mathcal{T} \;\; \exists! W_j \in \mathcal{W} : a_w(T_i) = W_j \tag{4.9}$$

- **Resource capacity:** In each isolation window $W_j \in \mathcal{W}$, at most c_k tasks are assigned to resource $R_k \in \mathcal{R}$:

$$\sum_{T_i \in \mathcal{T}: a_w(T_i) = W_j} \mathbb{1}_{[a_r(T_i) = R_k]} \leq c_k \quad \forall R_k \in \mathcal{R}, \forall W_j \in \mathcal{W} \tag{4.10}$$

- **Isolation window length:** Each isolation window is at least as long as the longest task assigned to it.

$$l_j \geq \max_{\substack{T_i \in \mathcal{T} \\ R_k \in \mathcal{R}}} \{p_{i,k} | a_w(T_i) = W_j \wedge a_r(T_i) = R_k\} \quad \forall W_j \in \mathcal{W} \tag{4.11}$$

- **Major frame length:** The combined length of all isolation windows is equal to the major frame length:

$$\sum_{W_j \in \mathcal{W}} l_j = h \tag{4.12}$$

Chapter 5

Solution approach

To find the optimal task mapping according to the PEWMS power model, we use ILP formalism. Section 5.1 details the ILP formulation of this problem. We also introduce heuristic solution methods as an alternative approach in Section 5.2 to gain a basis for comparison for evaluating the PEWMS model.

5.1 PEWMS model ILP formulation

We choose to use the ILP formalism as it poses several advantages for us. For example, we can use an existing ILP solver, which means that we do not have to spend time developing a custom algorithm for our problem. It is also much easier to alter the objective function and constraints of an ILP model than it is to modify a custom algorithm in case we need to make changes later. The biggest disadvantage of ILP is its possibly poor scalability, meaning that we might not be able to solve large problem instances.

The input for the model consists of:

- The set of tasks $\mathcal{T}$ with associated execution times $p_{i,k}$ and power consumption coefficients $a_{i,k}$ and $b_{i,k}$.
- The set of computing resources $\mathcal{R}$ with their capacities c_k.
- The major frame length h.

For each task $T_i \in \mathcal{T}$, resource $R_k \in \mathcal{R}$ and window $W_j \in \mathcal{W}$, we introduce binary variable $x_{i,j,k} \in \{0, 1\}$ modeling the assignment of task T_i to resource R_k and window W_j. Since the size of set $\mathcal{W}$ is not known in advance, we consider it to be the upper-bounded according to Equation (4.2). Next, we introduce a variable $l_j \in \mathbb{N}$ for each window $W_j \in \mathcal{W}$ representing the window length. Finally, we use auxiliary variable $B_{i,j,k} \in \mathbb{R}_0^+$ for each task $T_i \in \mathcal{T}$, resource $R_k \in \mathcal{R}$ and window $W_j \in \mathcal{W}$ to represent the increase in static power consumption of the system caused by task T_i being executed on resource R_k in window W_j.

The model, to which we will refer as global-ILP follows:

$$\min \frac{1}{h} \cdot \sum_{W_j \in \mathcal{W}} \left(\sum_{T_i \in \mathcal{T}} \sum_{R_k \in \mathcal{R}} x_{i,j,k} \cdot p_{i,k} \cdot a_{i,k} + \max_{\substack{T_i \in \mathcal{T} \\ R_k \in \mathcal{R}}} B_{i,j,k} \right) \quad \text{subject to:} \tag{5.1}$$

$$x_{i,j,k} = 1 \implies B_{i,j,k} = l_j \cdot b_{i,k} \quad \forall T_i \in \mathcal{T}, \forall W_j \in \mathcal{W}, \forall R_k \in \mathcal{R} \tag{5.2}$$

$$\sum_{W_j \in \mathcal{W}} \sum_{R_k \in \mathcal{R}} x_{i,j,k} = 1 \quad \forall T_i \in \mathcal{T} \tag{5.3}$$

$$\sum_{T_i \in \mathcal{T}} x_{i,j,k} \leq c_k \quad \forall W_j \in \mathcal{W}, \forall R_k \in \mathcal{R} \tag{5.4}$$

$$l_j \geq x_{i,j,k} \cdot p_{i,k} \quad \forall T_i \in \mathcal{T}, \forall W_j \in \mathcal{W}, \forall R_k \in \mathcal{R} \tag{5.5}$$

$$\sum_{W_j \in \mathcal{W}} l_j \leq h \tag{5.6}$$

Equation (5.1) is the objective function derived from our power model (Equation (4.7)). The aim is to minimize the average power consumption of the major frame in accordance with the problem statement presented in Section 4.3. The P_{idle} term was left out from the objective function as it is a constant that does not affect the quality of the solution. The value of $\frac{1}{h}$ is also a constant not affecting the quality of the solution, but we keep it in the objective function. This way, the value of the objective function can be interpreted as the estimated increase in average power consumption caused by the tasks being executed.

Implication (5.2) serves to assign a value to variable $B_{i,j,k}$. If $x_{i,j,k} = 0$, the value of $B_{i,j,k}$ is arbitrary and the solver will force it to zero, which is optimal.

Constraint (5.3) ensures that each task is assigned to exactly one resource and exactly one isolation window. Constraint (5.4) enforces the capacity limit of computing resources in each isolation window, and constraint (5.5) ensures that lengths of isolation windows are correctly set. Finally, constraint (5.6) ensures that the major frame length is not exceeded.

Since the order of windows is arbitrary, we use additional constraint forcing the ordering of the windows, which removes symmetrical solutions from the search space, thus significantly reducing the time needed to obtain a solution:

$$l_j \geq l_{j+1} \quad \forall j \in \{1, 2, ..., \ell - 1\} \tag{5.7}$$

There are two things to note about the global-ILP model. First thing is that the global-ILP model is not linear due to the maximum in the objective function (5.1) and the implication in constraint (5.2). While both of these expressions can be linearized by introducing additional variables and the use

of 'big M', modern ILP solvers can do the linearization internally. Therefore we leave the model in this form for better readability.

The second thing is that it does not output the empty window (and its length), which may be needed to satisfy the requirement (4.1). Because of this, constraint (5.6) uses inequality. However, it is simple to calculate the length of the empty window l_{empty} from the solution produced by global-ILP:

$$l_{empty} = h - \sum_{W_j \in \mathcal{W}} l_j \tag{5.8}$$

5.2 Heuristic solution methods

We adopt a decomposition approach for our heuristic solution methods to seek the resource mapping and the window mapping separately. Such approach is often used in literature, for example in [6] and [9]. A complete schedule will be found by first applying the resource assignment heuristic and then the window assignment heuristic.

Let us begin by introducing our window assignment heuristic. We call it the longest-tasks-first (LTF) schedule. LTF schedule, listed as Algorithm 1, works by assigning tasks to a window in non-increasing order of their execution times until all resources in the window are fully occupied. Then a new window is created, and this process is repeated. This heuristic method minimizes the total length of non-empty windows.[1] The assignment of tasks to windows is valid if the total length of non-empty windows does not exceed the major frame length. Therefore, if a feasible assignment of tasks to windows exists, it can be found by the LTF schedule method, which produces an assignment with the minimal total length of non-empty windows.

As for the resource assignment heuristic, we use the following resource assignment methods:

1. Random assignment of tasks to resources.
2. Assignment of tasks to resources that minimizes the total utilization of the system.
3. Assignment of tasks to resources based on a greedy heuristic proposed in literature.

There is not much to discuss about method 1, each task will simply be assigned to a resource randomly with a uniform probability. We will denote schedules with random resource assignment and a LTF schedule window assignment as RA+LTF. The details of methods 2 and 3 are described in Section 5.2.1 and Section 5.2.2 respectively.

[1] This scheduling problem corresponds to scheduling tasks on a bounded batching machine. Optimality of the LTF method for this problem was proven by Brucker et al. [12].

Algorithm 1: LTF-schedule heuristic

input : set of computational resources $\mathcal{R} = (R_1, ..., R_m)$, value c_k associated with each resource $R_k \in \mathcal{R}$ representing the resource capacity, set of tasks $\mathcal{T} = (T_1, ..., T_n)$, values p_i and r_i associated with each task $T_i \in \mathcal{T}$ representing execution time and index of resource the task has been assigned to respectively

output : value w_i associated with each task $T_i \in \mathcal{T}$ representing the index of the isolation window the task has been assigned to

```
1  u = ones(m) // vector of size m filled with ones
       capturing the index of the last window in which
       resource R_k is not fully occupied
2  i = zeroes(m) // vector of size m filled with zeroes
       capturing the used capacity of resource R_k in window
       u[k]
3  l = zeroes(n) // vector of size n filled with zeroes
       capturing lengths of non-empty isolation windows
4  sort T by p values in non-increasing order
5  foreach T_i ∈ T do
6      if i[r_i] == c[r_i] then
7          u[r_i] += 1
8          i[r_i] = 0
9      w_i = u[r_i]
10     i[r_i] += 1
11     l[u[r_i]] = max(p_i, l[u[r_i]])
```

5.2.1 Resource assignment minimizing the total system utilization

One of the possible uses for a resource assignment minimizing the total system utilization is to find a schedule to which more tasks can be added easily at a later time. Intuitively, one would also expect the resource assignment with minimal utilization not to heat up the processor as much as resource assignments with more utilization since it maximizes the processor idle time.

We define the total system utilization U for a given resource assignment as:

$$U = \frac{\sum_{R_k \in \mathcal{R}} \sum_{T_i \in \mathcal{T}: a_r(T_i) = R_k} p_{i,k}}{h \cdot \sum_{R_k \in \mathcal{R}} c_k} \tag{5.9}$$

To find the resource assignment minimizing U, we modify the global-ILP model by altering the objective function and dropping unneeded constraint (5.2). The modified model, henceforth referred to as minutil-ILP, follows:

$$\min \sum_{W_j \in \mathcal{W}} \sum_{T_i \in \mathcal{W}} \sum_{R_k \in \mathcal{R}} x_{i,j,k} \cdot p_{i,k} \quad \text{subject to:} \tag{5.10}$$

$$\sum_{W_j \in \mathcal{W}} \sum_{R_k \in \mathcal{R}} x_{i,j,k} = 1 \quad \forall T_i \in \mathcal{T} \tag{5.11}$$

$$\sum_{T_i \in \mathcal{T}} x_{i,j,k} \leq c_k \quad \forall W_j \in \mathcal{W}, \forall R_k \in \mathcal{R} \tag{5.12}$$

$$l_j \geq x_{i,j,k} \cdot p_{i,k} \quad \forall T_i \in \mathcal{T}, \forall W_j \in \mathcal{W}, \forall R_k \in \mathcal{R} \tag{5.13}$$

$$\sum_{W_j \in \mathcal{W}} l_j \leq h \tag{5.14}$$

$$l_j \geq l_{j+1} \quad \forall j \in \{1, 2, ..., \ell - 1\} \tag{5.15}$$

Equation (5.9) has been simplified for use in objective function (5.10) because task execution time, given by the assignment of a task to a resource, is the only variable present in that expression.

While minutil-ILP model maps tasks to both resources and to isolation windows, we disregard the window mapping as it is not deterministic and use LTF-schedule window assignment heuristic instead. We will refer to this method of obtaining a schedule as minutil+LTF.

5.2.2 Greedy assignment heuristic

The greedy heuristic we use is based on algorithms proposed by Zhou et al. [8] and Kuo et al. [9]. These algorithms share the idea of processing tasks in non-increasing order based on their projected power consumption behavior (though they differ in how this criterion is computed) and assigning the tasks to the best processor available.

To determine the order of processing tasks, we use the same method as Algorithm 1 proposed in [8], with the slight modification of disregarding the periodicity parameter of the tasks, as in our case, it is the same for all tasks and would not affect the ordering. Because the parameter μ_i of the referenced algorithm is not explained in detail, we identify it with our dynamic power consumption coefficient $a_{i,k}$ since the meaning should be similar. The task order we use therefore corresponds to the expected dynamic energy consumption of task T_i when executed on resource R_k:

$$E_{i,k} = a_{i,k} \cdot p_{i,k} \tag{5.16}$$

Next, we need to assign the task to a resource. We select the assignment with minimal $E_{i,k}$ value such that a feasible assignment of tasks to windows can be found. Determining if a resource assignment is feasible is easy in the referenced algorithms, as they do not work with isolation windows. Because we work with isolation windows, we have to resort to a more complicated approach. To determine if a feasible assignment of tasks to windows exists, we use a simplified version of the global-ILP model, without the objective function and unneeded constraint (5.2).

Let $T_{assigned}$ be a set of tasks for which the assignment to a resource has been decided. Let $r : \mathcal{T}_{assigned} \rightarrow \mathbb{N}$ be a function that maps the tasks from set $T_{assigned}$ to number k, representing the index of the resource task $T_i \in \mathcal{T}_{assigned}$ has been assigned to. For this model, we introduce additional constraint (5.23) fixing the already decided assignment of task T_i to resource R_k. The ILP model, from now on referred to as feasible-schedule-ILP, follows:

$$\min 0 \quad \text{subject to:} \tag{5.17}$$

$$\sum_{W_j \in \mathcal{W}} \sum_{R_k \in \mathcal{R}} x_{i,j,k} = 1 \quad \forall T_i \in \mathcal{T} \tag{5.18}$$

$$\sum_{T_i \in \mathcal{T}} x_{i,j,k} \leq c_k \quad \forall W_j \in \mathcal{W}, \forall R_k \in \mathcal{R} \tag{5.19}$$

$$l_j \geq x_{i,j,k} \cdot p_{i,k} \quad \forall T_i \in \mathcal{T}, \forall W_j \in \mathcal{W}, \forall R_k \in \mathcal{R} \tag{5.20}$$

$$\sum_{W_j \in \mathcal{W}} l_j \leq h \tag{5.21}$$

$$l_j \geq l_{j+1} \quad \forall j \in \{1, 2, ..., \ell - 1\} \tag{5.22}$$

$$\sum_{W_j \in \mathcal{W}} x_{i,j,r(T_i)} = 1 \quad \forall T_i \in \mathcal{T}_{assigned} \tag{5.23}$$

A feasible assignment of tasks $T_i \in \mathcal{T}$ to windows exists if and only if a feasible solution of the feasible-schedule-ILP model exists.

Algorithm 2 lists the complete pseudocode of the introduced heuristic algorithm, henceforth referred to as reference assignment heuristic.

Algorithm 2: Reference assignment heuristic

input : set of tasks $\mathcal{T}$, set of resources $\mathcal{R}$, major frame length h
output : assignment of tasks to resources

Function *feasibleWindowAssignmentExists($\mathcal{T}_{assigned}$)* **is**
    if *a feasible solution of feasible-schedule-ILP model for $\mathcal{T}_{assigned}$ exists* **then return** *true*;
    else return *false*;
Function *referenceAssignmentHeuristic()* **is**
    sort tasks in $\mathcal{T}$ by their maximal $E_{i,k}$ value in non-increasing order
    $\mathcal{T}_{assigned} = \{\ \}$
    foreach *task* $T_i \in \mathcal{T}$ **do**
        assigned = false
        sort possible resource assignments of T_i in non-decreasing order of $E_{i,k}$
        foreach *resource* R_k *to which* T_i *can be assigned* **do**
            assign T_i to R_k
            add T_i to $\mathcal{T}_{assigned}$
            if *feasibleWindowAssignmentExists($\mathcal{T}_{assigned}$)* **then**
                assigned = true
                break
            else
                unassign T_i from R_k
                remove T_i from $\mathcal{T}_{assigned}$
        if *assigned is false* **then**
            error: feasible assignment of tasks to resources does not exist
    return *resource assignment of tasks in* $\mathcal{T}_{assigned}$

Chapter 6

Software for preparing experiments

In the previous chapters we have introduced the problem of scheduling safety-critical tasks to isolation windows with the goal of minimizing average power consumption. To achieve this, we have created the global-ILP model. The next step is to evaluate the model on real hardware and compare it against other methods introduced in Chapter 5. This means that we need to create several problem instances (sets of tasks). For each instance, we will create schedules to be evaluated by applying the aforementioned methods. We will then execute these schedules on our testbed and evaluate their results.

The MEK board in our testbed is powered by a Linux-based system. The Linux kernel does not natively support scheduling of tasks in the way we need, however, it does provide means to achieve it with the help of additional tools. DEmOS [13] is an open-source tool developed specifically for this reason. It is able to execute tasks on specified CPU cores and limit their execution to specified time intervals. DEmOS can be configured via YAML files allowing us to specify how DEmOS will execute the tasks, i.e. the assignment to CPU cores and the time intervals corresponding to the isolation windows.

However, the path toward execution of a schedule is not as simple as creating a configuration for DEmOS. We consider the input to be the list of benchmarks that can be run on the system. For each of these benchmarks, we know the characteristics for the available computing clusters: execution time for a single iteration and the power consumption coefficients. From this list of benchmarks, we need to create test instances (select a given number of benchmarks) and for these instances, we have to create the actual schedules. Finally, a configuration file for DEmOS must be produced in order to execute the schedules to evaluate them. Since performing these steps by hand would be extremely time consuming, we will create support software to automate the workflow.

6.1 Analysis

The motivation for creating software that will help with preparing the experiments to evaluate the PEWMS power model is clear. We will now discuss in detail what tasks we require the software to perform.

The support software shall cover the path between the input data and

the configuration for DEmOS. This path can be broken up into three steps: creating a problem instance, creating schedules for the instance, and creating a DEmOS configuration.

Let us discuss the input data first. The input shall be composed of two parts: a description of the hardware platform and characteristics of available benchmarks. The benchmark characteristics shall consist of single iteration execution times and power consumption characteristics (the dynamic and static power consumption coefficients according to the PEWMS model) for available computing clusters. The description of the hardware platform shall consist of the available computing clusters and their capacities. This data is all the information needed to create a problem instance and solve it, that is, to create the schedules.

The next step entails the creation of the schedule. We consider the following methods of creating a schedule:

1. The global-ILP model.
2. Random resource assignment with LTF-schedule.
3. minutil-ILP resource assignment with LTF-schedule.
4. Resource assignment given by the reference assignment heuristic with LTF-schedule.

It is apparent from this list that for three out of the four methods considered (2, 3, 4), the resource assignment and window assignment phases are split. Thus it would make sense for our software to consider the resource assignment and window assignment part of the problem to be separate so that the LTF-schedule algorithm does not have to be reimplemented each time. The exception is method 1, where the schedule can be obtained directly by solving the global-ILP model.

Another thing to note is that three out of the four methods (1, 3, 4) require an ILP model to be solved. As solving an ILP model efficiently is a challenging task, it shall be delegated to an existing ILP solver.

The final step consists of transforming the internal representation of the produced schedule to a configuration for DEmOS. Another useful feature would be a visualization of the produced schedule to give us an idea of what will actually be executed on the target system. It might also be useful for the internal representation to be saved alongside the DEmOS configuration. This representation should be human-readable as well as machine-readable (format such as JSON, YAML, or XML). This would give us the ability to process the solution in other software to for example analyze some properties of the schedule, such as resource utilization or window fullness. The software should also allow for the representation to be converted to DEmOS configuration independently of creating and solving an instance to allow us to make changes in the intermediate representation. The discussed pipeline for producing and solving problem instances is shown in Figure 6.1.

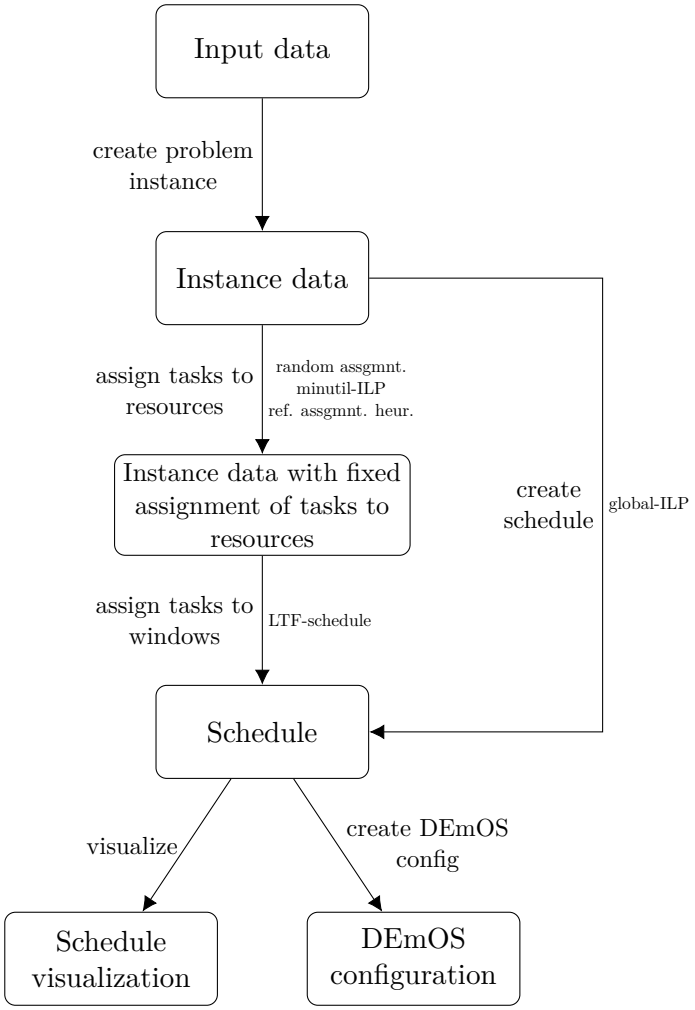


Figure 6.1: Problem instance pipeline

6.1.1 Software requirements

Based on the conducted analysis, we summarize the requirements for the support software to be created. We require that the support software we shall create:

- Is able to create a problem instance so that schedules to be evaluated can be produced.
- Is able to solve the global-ILP model with the help of an ILP solver so that schedules produced by this method can be evaluated.
- Is able to solve the minutil-ILP model with the help of an ILP solver so that schedules with this resource assignment method can be evaluated.
- Is able to execute the reference assignment heuristic so that schedules with this resource assignment method can be evaluated.

- Is able to solve the feasible-assignment-ILP model with the help of an ILP solver so that it is able to execute the reference assignment heuristic.
- Is able to randomly assign tasks to resources so that schedules with this resource assignment method can be evaluated.
- Is able to execute the LTF-schedule heuristic so that it can produce a schedule for methods that only produce resource assignments.
- Is able to export the solution to a DEmOS configuration so that the schedule can be executed on our testbed.
- Is able to visualize the produced schedule so that we know how the schedule looks like.
- Is able to save the solution of a problem instance in a format that is both human and machine-readable, so that it can be integrated with other software as well as inspected and manipulated by us, should we want to.

6.2 Design

By analyzing the requirements we have for the support software, we discovered that we require it to perform a multitude of varied tasks: generating problem instances, solving ILP models, executing heuristic algorithms, and more. While it was not explicitly specified as a requirement, it might be useful to be able to save intermediate results between the stages shown in Figure 6.1. A simple way to design and implement software that fits all of these requirements would be to adopt the Unix philosophy [14]. Namely, instead of making a single program that 'does it all', we will instead create several small and simple programs, each designed for a single specific task. We will achieve the desired results by chaining these small programs together. That implies that we will need to design a way of exchanging the data between the programs which gives us a way of saving the output of each stage of the data pipeline.

We shall create a program for each of the transitions in Figure 6.1:

- Instance generator: will use the input data to create a problem instance.
- Solver for global-ILP: will use the output of the instance generator to produce a complete schedule by solving the global-ILP model.
- Resource assignment solver: will use the output of instance generator to assign tasks to resources either by solving the minutil-ILP model, by executing the reference assignment heuristic (which includes solving the feasible-assignment-ILP model) or by assigning tasks to resources randomly.
- Window assignment solver: will use a fixed resource assignment to create a schedule by executing the LTF-schedule heuristic.

- DEmOS configuration exporter: will export the completed schedule to a configuration for DEmOS.
- Visualizer: will create a visual representation of the schedule in the form of a bitmap image.

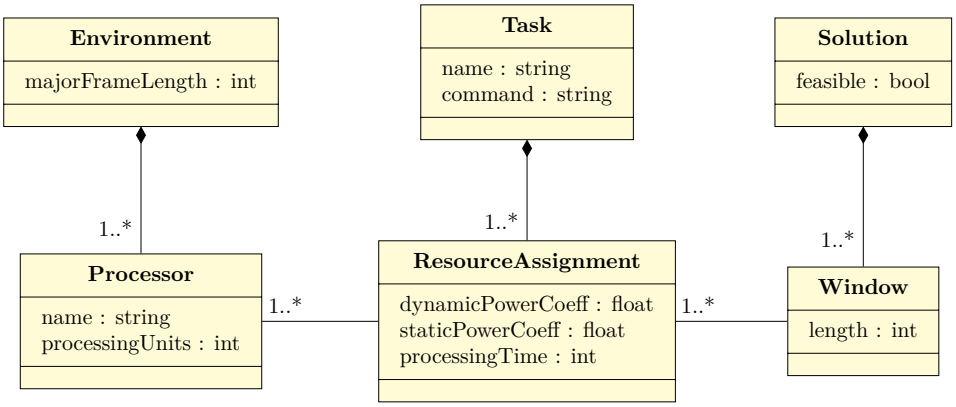


Figure 6.2: Conceptual data model of our software

Figure 6.2 shows the conceptual data model of the designed software. This is the data that will be exchanged between the individual programs in our toolchain. The model can be divided into three parts, which can be thought of as layers: the *environment*, the *tasks*, and the *solution*. Each step of the data pipeline adds another layer of data. We start with a description of the environment consisting of a list of available computing resources (computing clusters) and the major frame length. This will be passed to the instance generator along with the characteristics of available benchmarks. Note that the benchmark data is not part of the data model as it will be used only by the instance generator. We expect that this data will be provided in a CSV file. The instance generator shall also be able to accept additional parameters, namely the desired size of the problem instance (number of tasks) and the desired range of execution time to output. The parameters for generating the problem instance shall be passed as the program arguments.

The instance generator will use the provided inputs to add a second layer of data: the tasks. Each task has a name (for easy identification when visualizing the schedule), a command for running the task, and a list of possible resource assignments.

The solvers will use the list of tasks to add the final data layer: the solution. The solution consists of an attribute identifying its feasibility. A feasible solution also contains a list of isolation windows, where each window has a length and a list of task assignments (describing the assignment of a task to a resource). The execution order of the windows is given by their order in this list.

We skipped over the 'assign tasks to resources' transition because no actual data is added in this step, as it consists only of selecting a single resource assignment for each task. Nevertheless, this information still needs to be

passed from the resource assignment solver to the window assignment solver. We shall address this point during implementation.

6.3 Implementation

We implemented the following six tools in accordance with the design described in Section 6.2: *instance_generator*, *ilp_global_solver*, *assignment_solver*, *schedule_solver*, *demos_config_export*, and *visualizer*. The implementation was done in C++, as it is a highly portable language with great performance and an extensive standard library. We chose JSON as the format for data exchange between individual tools. JSON is a lightweight format that is human-readable and is supported in all common programming languages, either directly in the language's standard library or by an external library. As C++ does not natively support the JSON format, we used the *json* library created by *nlohmann* [15] available under a free license.

As we have previously mentioned, we will use an existing ILP solver for the purpose of solving our ILP models. This affects the *ilp_global_solver* and *assignment_solver* tools. We chose to use the Gurobi [16] solver, because it offers good performance, academic license, and we already have experience using it.

Consistent with the Unix philosophy, our tools read from the standard input and write to the standard output by default. This allows for them to be chained together by Unix pipes. The use of the Gurobi solver introduces a small issue to this system. By default, Gurobi logs its progress to the standard output, which interferes with the expected output of the tool in question. In case this presents an issue, it is possible to change the configuration of Gurobi to disable its output. This can be done via *gurobi.env* file present in the working directory. Configuration of Gurobi in this way is not limited to its output, other useful parameters such as the number of threads to use while solving an ILP model or the time limit for finding a solution can be set, for example:

```
OutputFlag 0
Threads 8
TimeLimit 60
```

This configuration disables the output of Gurobi, specifies that 8 threads will be used while solving the model, and sets the time limit for finding a solution to 60 seconds.

If a time limit is set, the tools making use of ILP models will use the best solution available once the time limit expires. If the use of standard input/output is not desired, the behavior of our tools can be altered through program parameters to use files instead.

Let us now discuss the implementation of the data model. Some minor changes (mostly additions) were made to make the software more flexible for use in future research. However, the core idea stayed the same. At

the start of the toolchain pipeline, a JSON describing the environment is provided. Each tool then adds another layer of data. These layers are realized as JSON objects. There are four types of these objects: *environment*, *assignmentCharacteristic*, *task*, and *solution*. At each step of the pipeline, the JSON representing the intermediate result contains a subset of these objects. Note that the *assignmentCharacteristic* object is not required in the stages after being processed by *ilp_global_solver* or *assignment_solver*, but since there is no reason to remove it from the file, it is kept there. Thus, if the file went through the entire pipeline, it will contain all of the mentioned objects at the end of it. Listing 1 shows such JSON.

```
{
    "environment":{
        "majorFrameLength":2000,
        "problemVersion":1,
        "processors":[
            { "name":"A53", "processingUnits":4, "type":"main_processor" },
            { "name":"A72", "processingUnits":2, "type":"main_processor" } ]
    },
    "assignmentCharacteristics":[
        { "task":"T1", "command":"yes >/dev/null", "resourceAssignments":[
            { "slope":0.2, "intercept":0.3, "length":100,
                 "processors": [{"processingUnits":1, "processor":"A72"}]
            }] },
        { "task":"T2", "command":"yes >/dev/null", "resourceAssignments":[
            { "slope":0.1, "intercept":0.2, "length":80,
                "processors": [{"processingUnits":1, "processor":"A53"}]
            }] }
    ],
    "tasks":[
        { "assignmentIndex":0, "command":"yes >/dev/null", "length":100,
            "name":"T1", "processors": [{"processingUnits":1,
            "processor":"A72"}]},
        { "assignmentIndex":0, "command":"yes >/dev/null", "length":80,
            "name":"T2", "processors": [{"processingUnits":1,
            "processor":"A53"}] }
    ],
    "solution":{
        "feasible":true,
        "solutionTime":10,
        "solverMetadata":{},
        "solverName":"global-ILP Solver",
        "windows":[
            { "length":100, "tasks":[{"length":100, "processingUnit":0,
                "processor":"A72", "start":0, "task":"T1"}] },
            { "length":80, "tasks":[{"length":80, "processingUnit":0,
                "processor":"A53", "start":0, "task":"T2"}] } ]
    }
}
```

Listing 1: Complete JSON representation of a solved instance

The notable differences from the conceptual model include:

- The *environment* object contains a new field *problemVersion*, which was added to help differentiate between different formats of input files used during the development of the software.

- The *processor* object inside the *environment* object contains a new field *processorType*, which was added to identify the different types of computing resources (CPU and GPU). It is unused for the purposes of this thesis.
- The *assignmentCharacteristic* object corresponds to the Task object defined in the conceptual model. A new object type stored in the *tasks* array was added. The objects in this array represent the selected task assignment and contain all necessary data related to it. This was done to simplify subsequent processing of the data, which is needed in the window assignment solver as well as in the visualizer and the configuration exporter for DEmOS.
- The dynamic power consumption coefficient is named *slope* and the static power consumption coefficient is named *intercept*.
- The *solution* object contains metadata provided by the solver that are intended mainly for statistics and analyzing the solver's performance.

The data pipeline as realized by our toolchain is shown in Figure 6.3.

The next thing to address is the second part of input data for *instance_generator*, which is the benchmark data. We mentioned that we expect it to be a CSV file. The expected format of the file is as follows:

```
benchmark,affinity,slope,intercept,runtime
dijkstra,A53,0.23314,0.21280,0.01750
dijkstra,A72,0.20275,0.22218,0.00033
```

This example shows the characteristics of the *dijkstra* benchmark when executed on the A53 and the A72 computing clusters.

It is also possible to add a column *command* with a string containing the command to be executed for that particular benchmark. If this column is not provided, the generator will use the benchmark name as the value of the command field. The user can then use a tool such as *sed* to replace these values as required.

The *visualizer* tool makes use of the *CImg* [17] and the *lodepng* [18] libraries for creating a bitmap image visualizing the schedule. Both of these libraries are available under a free license. A visualized schedule produced by the *visualizer* tool can be seen in Figure 6.4.

Despite none of our tools using any platform-specific code, their portability is limited by the use of the Gurobi ILP solver. Nonetheless, they have been tested on recent versions of both GNU/Linux and Windows operating systems. The source code of the software, as well as the instructions for compiling and using it can be found on the attached CD.

6.4 Testing

Because the software we created consists of simple tools built for a single purpose, formal testing was not conducted. Rather, we used exploratory

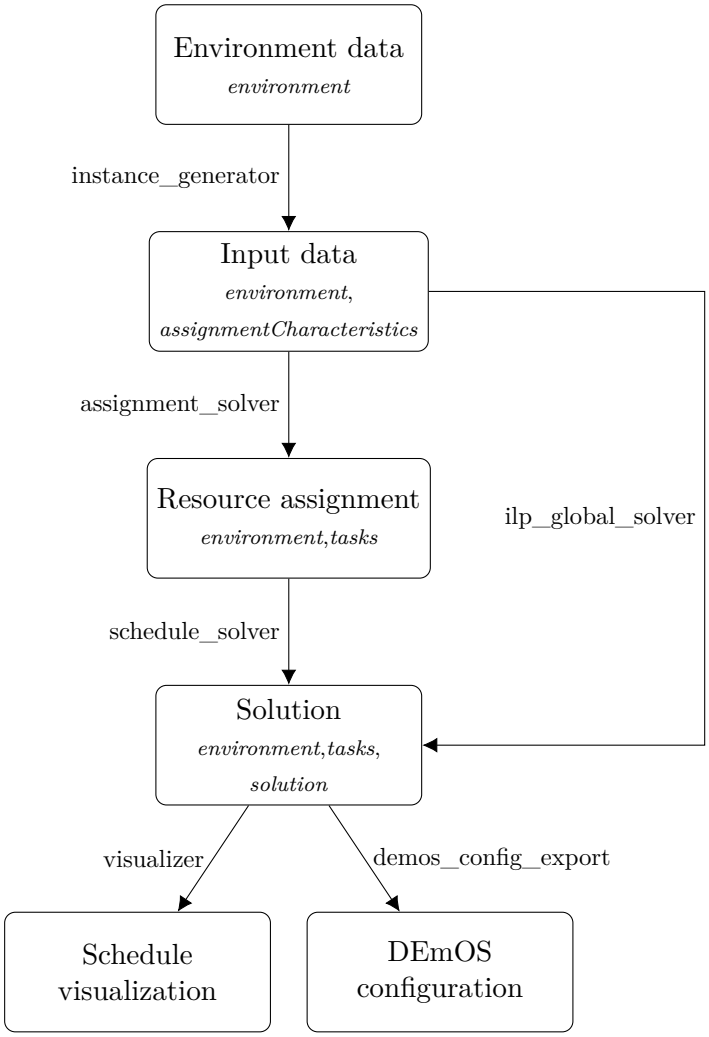


Figure 6.3: Realization of the problem instance pipeline

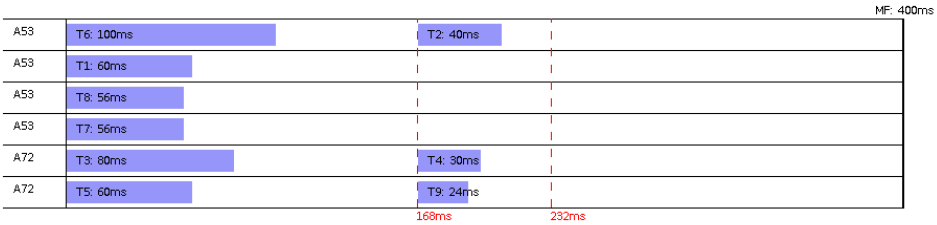


Figure 6.4: Schedule visualization produced by the visualizer tool

testing to verify that each tool is working correctly. Further, using the tools to prepare the actual experiments used for evaluating the PEWMS model served as user acceptance testing. It led to a discovery and remediation of some minor usability issues.

Chapter 7

Experimental evaluation

To evaluate the PEWMS power model, we generated a number of test instances. For these instances, we created schedules according to the solution methods introduced in Chapter 5. Details of how the experiments were set up are explained in Section 7.1. We executed the schedules on our testbed (described in Section 2.1) and measured the steady-state temperatures and average power consumptions. The results are reported in Section 7.2 and Section 7.3 respectively and summarized in Section 7.4.

7.1 Experiment setup

The workload for our test instances consisted of six selected benchmarks from the Taclebench benchmark suite [19] (*dijkstra*, *fft*, *prime*, *sha*, *susan*, *test3*), a 3D rendering tool *tinyrenderer* [20], and a memory stressing benchmark *membench* [21]. The membench tool stresses the memory by continuously performing read and write operations over a fixed size data. We used the tool in two different configurations: *membench-1M* (using 1MB of data), which was intended for stressing the L2 cache, and *membench-4M* (using 4MB of data), intended for stressing the main memory. The execution times and power consumption coefficient of the benchmarks are listed in Table 7.1.

	Single iteration execution time			Dynamic power consumption coeff.		Static power consumption coeff.	
Benchmark	**A53**	**A72**	**ratio**	**A53**	**A72**	**A53**	**A72**
dijkstra	17.50 ms	10.80 ms	1.62	0.233	0.914	0.213	0.211
fft	0.332 ms	0.114 ms	2.91	0.203	1.201	0.222	0.231
prime	0.372 µs	0.199 µs	1.87	0.208	1.091	0.219	0.141
sha	0.454 ms	0.170 ms	2.67	0.276	1.467	0.224	0.160
susan	20.49 ms	5.64 ms	3.63	0.176	1.220	0.233	0.175
test3	67.14 ms	34.77 ms	1.93	0.343	1.298	0.248	0.184
tinyrenderer	953 ms	409 ms	2.33	0.297	1.279	0.963	0.925
membench-1M	814 ms	606 ms	1.34	0.528	0.755	0.547	1.263
membench-4M	7.15 s	5.45 s	1.31	0.287	0.559	1.544	1.593

Table 7.1: Benchmark parameters

We generated six problem instances: four without the membench bench-

marks and two with them (Instance #4 and #5). Each generated instance consisted of 25 tasks randomly selected (with repetitions and uniform probability) from the benchmark set. A random execution time from interval $[40, 160]$ ms was selected for execution on the A72 computing cluster for each task. The execution time for the A53 cluster was scaled accordingly, based on the ratio between execution times of a single iteration of the benchmark. These times were chosen to get major frames with lengths of around one second, as according to our industrial partner, such major frame lengths are typical.

The major frame length was set as $h = \frac{\bar{p} \cdot n}{\kappa}$, where $n = 25$ (the number of tasks), $\bar{p}$ is the average task execution time across all clusters and $\kappa = 3.5$ is an empirical constant setting the major frame length not too tight for the instance to be infeasible, but not too loose to have a trivial solution.

Each schedule was executed three times to measure the uncertainty of the results. Each time, the experiment ran for 30 minutes, which was enough time for the system to reach the steady-state temperature. After each run, the system was allowed to cool down to a base temperature of 30 °C, which could take up to six minutes depending on the ambient temperature. This resulted in over 54 hours needed in total for the experiments to be completed. The time-demanding nature of these experiments is the main reason why we only present six different instances.

The frequency of the A53 computing cluster was set to 1200 MHz. The frequency of the A72 cluster was set to 1596 MHz. Both of these are the maximal available frequencies for each cluster. The system's idle power was measured to be $P_{idle} = 5.59$ W.

7.2 Thermal results

In Table 7.2, we report the average relative steady-state temperature T_{rel} across all three runs of the schedule and the standard deviation for the different schedules. The relative steady-state temperature T_{rel} is calculated as:

$$T_{rel} = T_{\infty} - T_{amb} \tag{7.1}$$

where T_{∞} is the steady-state temperature as measured by the temperature sensor in the i.MX8 chip (we used the thermal zone 0 sensor for all measurements), and T_{amb} is the ambient temperature obtained from the ambient temperature sensor. The results from Table 7.2 are visualized in Figure 7.1.

The global-ILP schedules show the best relative steady-state temperature results, being on average 12.63% better when compared against minutil+LTF schedules and by 7.38% better when compared against the random resource assignment schedules.

The schedules based on the reference assignment heuristic are on average 9.50% better than minutil+LTF schedules and by 4.08% better than random resource assignment schedules. However, they are by 3.44% worse on average than the global-ILP schedules.

Instance	Average T_{rel} [°C]					
	global-ILP	minutil+LTF	RA+LTF 1	RA+LTF 2	RA+LTF 3	Ref. assgmnt. heuristic+LTF
#1	29.32 ± 0.18	33.29 ± 0.10	31.81 ± 0.20	32.08 ± 0.23	31.91 ± 0.25	30.77 ± 0.20
#2	29.60 ± 0.36	34.25 ± 0.21	32.25 ± 0.06	31.97 ± 0.13	31.14 ± 0.25	31.16 ± 0.28
#3	28.96 ± 0.44	32.78 ± 0.61	30.51 ± 0.11	30.33 ± 0.25	30.71 ± 0.28	29.54 ± 0.49
#4	29.38 ± 0.27	33.04 ± 0.40	31.03 ± 0.26	31.76 ± 0.26	29.96 ± 0.20	29.68 ± 0.29
#5	29.29 ± 0.38	33.94 ± 0.16	32.87 ± 0.32	33.56 ± 0.23	32.24 ± 0.16	30.36 ± 0.13
#6	29.38 ± 0.29	34.09 ± 0.05	31.86 ± 0.30	31.96 ± 0.45	32.27 ± 0.17	30.74 ± 0.11

Table 7.2: Average relative steady-state temperatures of the evaluated schedules

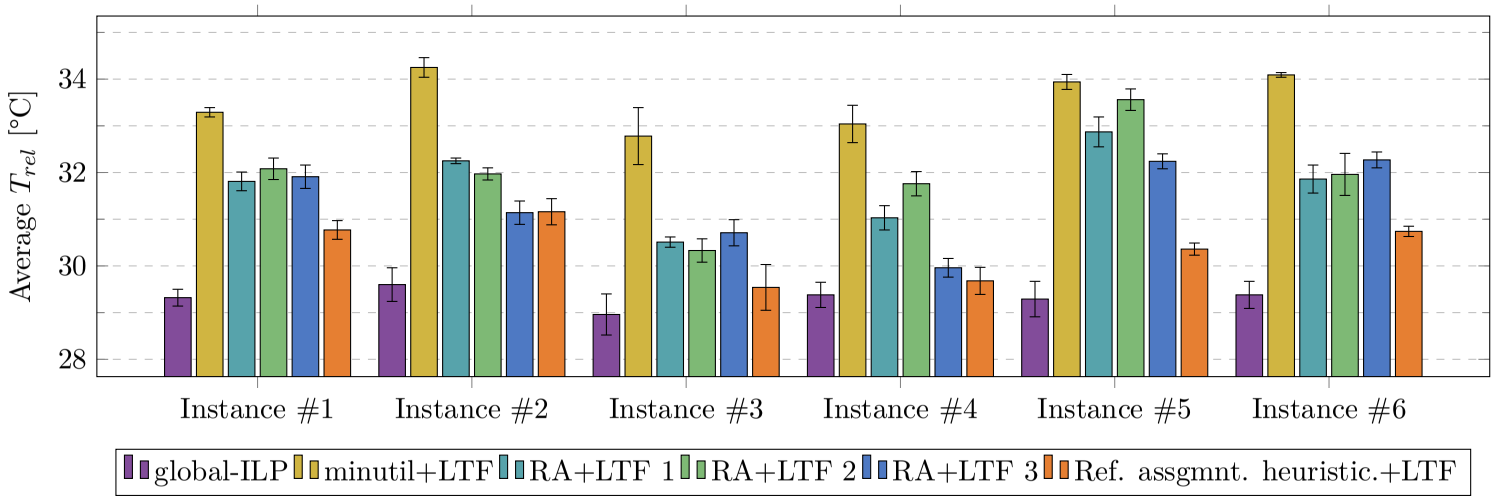


Figure 7.1: Comparison of the average relative steady-state temperatures of the evaluated schedules

The reference assignment heuristic tends to assign more tasks to the A72 cores, though they are usually short. Combined with the use of the LTF-schedule heuristic window assignment, this results in most of the isolation windows being fully occupied. Compare, for example, the global-ILP schedule for Instance #1 shown in Figure 7.2 with the reference assignment heuristic+LTF schedule for the same instance shown in Figure 7.3.

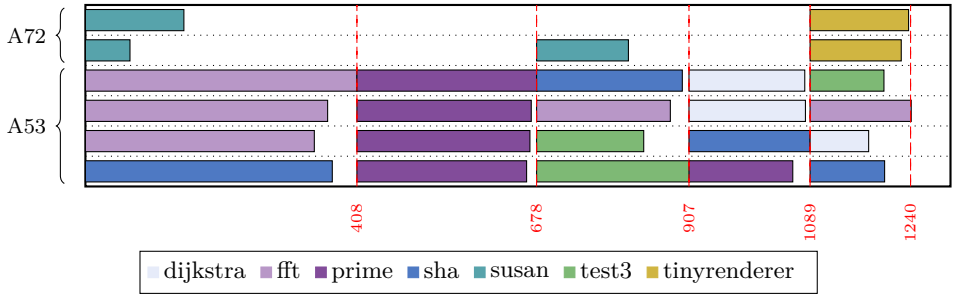


Figure 7.2: global-ILP schedule for Instance #1

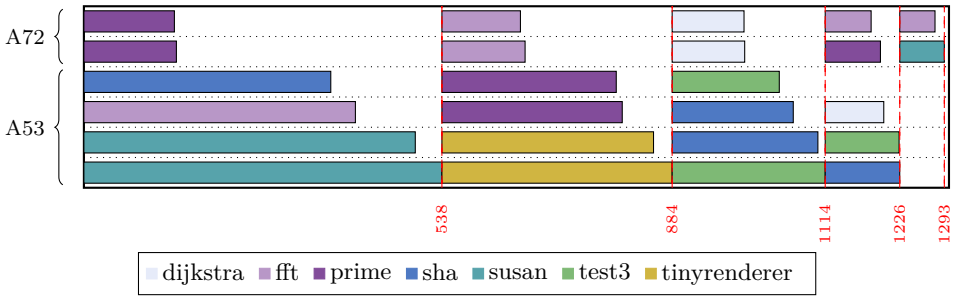


Figure 7.3: Reference assignment heuristic+LTF schedule for Instance #1

The minutil+LTF schedules consistently show the worst thermal results out of the evaluated methods. This can be attributed to the fact that this method prefers the more powerful A72 cores to use the shortest task execution times possible. By doing so, it minimizes the total system utilization. The A72 cores, however, heat up significantly more than the A53 cores, which results in bad thermal performance. An example minutil+LTF can be seen in Figure 7.4, which shows the minutil+LTF schedule for Instance #1.

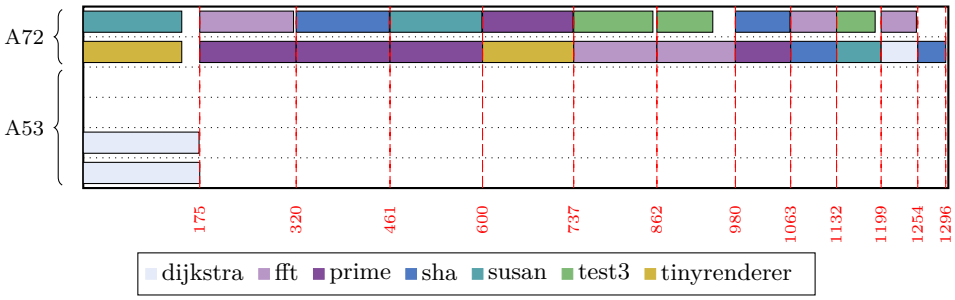


Figure 7.4: minutil+LTF schedule for Instance #1

7.3 Power estimation results

Besides thermal results, we are also interested in evaluating the accuracy of the average power consumption estimation of the PEWMS model since estimating the system's average power consumption is its primary purpose. To evaluate the accuracy, we define the power estimation error ϵ as:

$$\epsilon = \frac{P_{measured} - P_{estimated}}{P_{estimated}} \cdot 100 \tag{7.2}$$

where $P_{measured}$ is the average power consumption of the system during a run and $P_{estimated}$ is the average power consumption as estimated by the PEWMS model.

In Table 7.3 we report the power estimation error ϵ for the different schedules evaluated. To calculate the power estimation error, we use the average of $P_{measured}$ across all three runs. Note that the standard deviation of the measured power consumptions does not exceed 0.05 W for all except three of the evaluated schedules.

	Power estimation error ϵ [%]					
Instance	global-ILP	minutil+LTF	RA+LTF 1	RA+LTF 2	RA+LTF 3	Ref. assgmnt. heuristic+LTF
#1	-0.09	-0.53	-0.13	-1.69	-1.60	-0.16
#2	0.15	0.94	0.06	-0.26	-2.73	-0.33
#3	-1.70	-0.32	-2.23	2.72	-0.56	-2.66
#4	1.50	0.60	-0.66	1.24	-1.24	0.95
#5	-0.42	-0.64	0.90	0.64	-0.51	-7.69
#6	-0.24	0.59	-2.87	-1.04	-1.05	-2.88

Table 7.3: Power estimation errors of the PEWMS model for the evaluated schedules

The power estimation error is -0.66% on average, meaning that the PEWMS power model tends to overestimate the power consumption, which we expected (see Section 4.2). The greatest estimation error of -7.69% (Instance #5, reference assignment heuristic+LTF schedule) is the only occurrence of error greater than 5%. Figure 7.5 shows the schedule in question. A possible explanation for the error could be the presence of the membench benchmarks in four windows. Because the membench tool is designed to stress memory, when executed in parallel with other tasks, a congestion occurs as the tasks being executed compete for the L2 cache, which alters the power consumption behavior of the system. This effect is even more pronounced if multiple membench instances are running in parallel, which occurs in two of the windows in this particular schedule. Still, we consider the estimation errors of the PEWMS power model to be within an acceptable range.

7.4 Summary of results

The performed experiments showed that the global-ILP schedules consistently exhibited the best thermal results: these schedules achieved on average

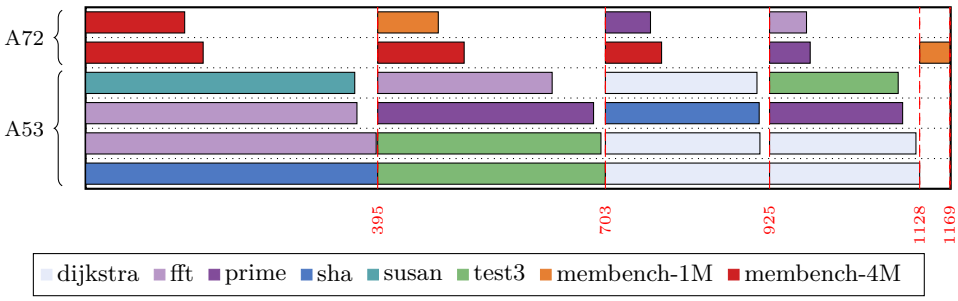


Figure 7.5: Reference assignment heuristic+LTF schedule for Instance #5

12.63% better results than minutil+LTF schedules, 7.38% better results than schedules based on random resource assignment, and 3.44% better results than reference assignment heuristic schedules. A visualization of the evaluated schedules can be found in Appendix D.

The average power estimation error of the PEWMS power model was -0.66%. The model had a tendency to overestimate the power consumption, which conforms to our expectations. The worst estimation error the model showed was -7.69%. This was an outlier case and a single occurrence of error greater than 5%, which could be explained by the presence of a memory stressing benchmark. With these numbers in mind, we conclude that the PEWMS power model is able to estimate the average power consumption of our system reasonably well. We provide the measured and estimated average power consumption values in Appendix E.

Only six different problem instances were used for the evaluation because of the time-demanding nature of the experiments. At the time of writing, more experiments with a more varied task set are taking place. The preliminary results of these experiments further support our findings: the global-ILP schedules still achieve the best thermal results, and the average estimation error of the PEWMS model remains inside a 5% range. More details of the preliminary results are contained in Appendix F.

Several authors [6, 7] have noted that complex ILP models can suffer from performance issues when solving larger problem instances. We have encountered this issue as well, as we were not able to find an optimal solution for instances with more than 32 tasks within a reasonable timeframe. Our assessment of the global-ILP model scalability can be found in Appendix G.

Chapter 8

Major frame length/temperature tradeoff

Sheikh et al. [4] postulate that in the case of thermal-aware scheduling, there exists a tradeoff between schedule length and steady-state temperature. For us, this would imply a tradeoff between the major frame length and the relative steady-state temperature of our system. We evaluate the tradeoff by conducting the following experiment: we take a problem instance used for evaluating the PEWMS power model and create several schedules for this instance with differing major frame lengths. We use the global-ILP and minutil+LTF methods for creating schedules as the best and the worst performing schedules according to their thermal results to evaluate the tradeoff regardless of schedule quality.

We make an arbitrary decision to use Instance #2 from Chapter 7, for which we use major frame lengths $h = \{930, 1130, 1330, 1530, 1730, 1930\}$ ms. For shorter major frames, the problem instance is no longer feasible and for longer major frames, no further improvement to the global-ILP optimization objective can be achieved. For each of these major frame lengths, we create optimal global-ILP and minutil+LTF schedules. We also evaluate the global-ILP schedule for $h = 930$ ms in the following way: for the different major frame lengths, we keep the resource and window assignments and only extend the length of the empty window to see how only adding idle time to the schedule affects it.

8.1 Thermal results

The described experiment was conducted under the same conditions as described in Section 7.1. Three runs of each schedule were performed and the steady-state temperatures measured. The average of measured temperatures is listed in Table 8.1 and shown in a graph in Figure 8.1.

The results clearly show that by increasing the major frame length, the steady-state temperature of the system decreases. This can be attributed to more time being available to perform the same amount of work. The global-ILP model can find a more efficient task mapping, while the other schedules utilize the system less, giving it more time to cool down. In our case, by approximately doubling the major frame length, the steady-state temperature of the schedules produced by the global-ILP method decreased

h [ms]	global-ILP	Average T_{rel} [°C] global-ILP solution for $h = 930$	minutil+LTF
930	35.90 ± 0.29	35.90 ± 0.09	36.94 ± 0.16
1130	31.88 ± 0.19	33.06 ± 0.33	36.01 ± 0.16
1330	29.60 ± 0.36	31.35 ± 0.14	34.30 ± 0.19
1530	28.33 ± 0.32	29.92 ± 0.25	32.61 ± 0.06
1730	27.09 ± 0.31	28.90 ± 0.15	31.30 ± 0.08
1930	25.99 ± 0.27	28.02 ± 0.29	30.21 ± 0.16

Table 8.1: Average relative steady-state temperatures of the schedules with different major frame lengths

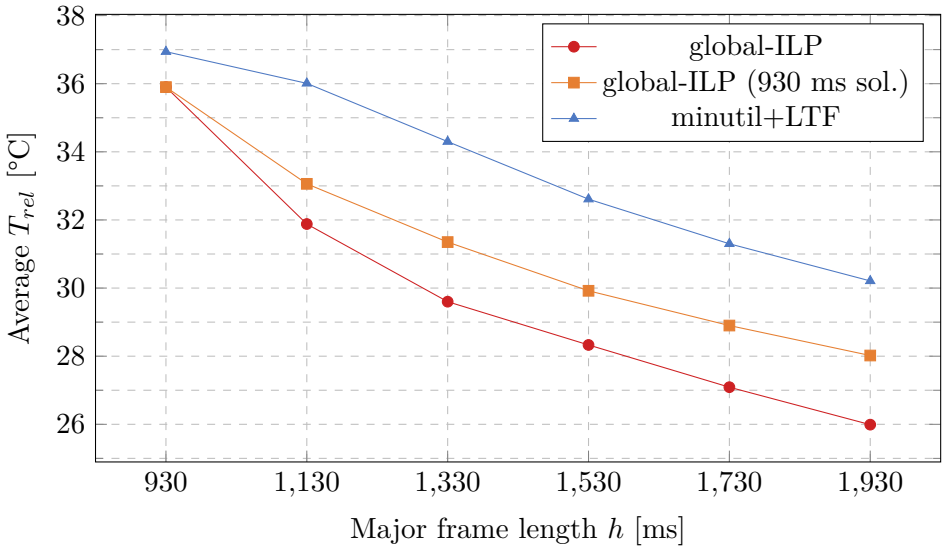


Figure 8.1: Visualization of the average relative steady-state temperatures of the schedules with different major frame lengths

by 27%, and by 18% in case of the minutil+LTF schedules.

The greatest decrease in temperature can be achieved on tight schedules. By extending the major frame length from 930 ms to 1130 ms, the global-ILP schedule temperature improved by 4 °C. However, when the same amount of time was added to the global-ILP schedule for $h = 1730$ ms, the steady-state temperature improved only by 1.1 °C.

Adding idle time to the global-ILP solution for $h = 930$ ms also served to improve the steady-state temperature, however, not by as much as finding the optimal schedule. By comparing the global-ILP schedule for $h = 1930$ ms (shown in Figure 8.2) with the global-ILP schedule for $h = 930$ ms with idle time added (shown in Figure 8.3), an interesting observation can be made. The total resource utilization of the optimal global-ILP schedule for $h = 1930$ ms is greater ($U = 55.5\%$) than that of the global-ILP solution for $h = 930$ ms with idle time added ($U = 36.5\%$), despite the steady-state temperature of the optimal schedule being lower. This result can be attributed

to A53 cores consuming less power and consequently heating up less than the A72 cores, at the cost of them having worse computational performance. The worse computational performance results in more time needed to complete the tasks, which causes the higher resource utilization.

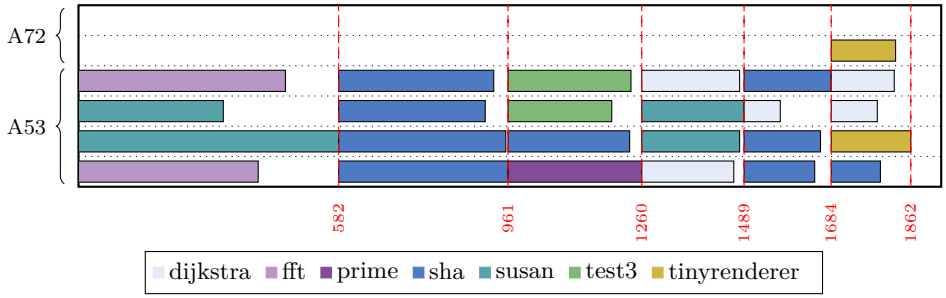


Figure 8.2: global-ILP schedule for $h = 1930$ ms

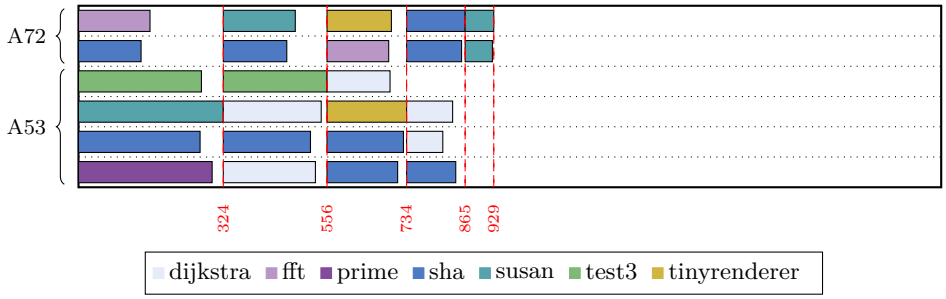


Figure 8.3: global-ILP schedule for $h = 930$ ms with 1000 ms of idle time added

8.2 Power estimation results

For the sake of completeness, we measured the average power consumption of the schedules used for evaluating the tradeoff between major frame length and steady-state temperature. Table 8.2 shows the power estimation error of the PEWMS model for these schedules.

	Power estimation error ϵ [%]		
h [ms]	global-ILP	global-ILP solution for $h = 930$	minutil+LTF
930	1.62	2.28	1.57
1130	0.64	1.24	1.20
1330	0.15	1.57	0.98
1530	-0.05	1.15	0.79
1730	-0.44	1.15	0.81
1930	-1.03	1.06	0.68

Table 8.2: Power estimation errors of the PEWMS model for the schedules with different major frame lengths

In this experiment, the PEWMS model underestimated the power consumption of most of the schedules. Still, with an average estimation error of 0.85% and the greatest estimation error being 2.28%, we consider the accuracy of the PEWMS model to be reasonably good.

Chapter 9

Conclusion

The main goal of this thesis was to design, implement and evaluate an offline thermal-aware scheduling method for periodic safety-critical workloads on a system with heterogeneous architecture. Our focus was on mapping tasks to different computing clusters available on an MPSoC-based system to reduce its steady-state temperature. Instead of minimizing the steady-state temperature directly, we worked on minimizing the average power consumption of the system since there is a direct relationship between the power consumption and the steady-state temperature of the system.

With the use of empirical observations and measurements, we created the PEWMS power estimation model. Before validating the model, we had to design and implement software tools to help us prepare the experiments. The subsequent evaluation of the schedules based on the PEWMS model confirmed the validity of the approach used, as these schedules exhibited steady-state temperatures lower by 3% to 12% compared to other methods we tested in the main evaluation experiment presented in Chapter 7.

We also verified the accuracy of the PEWMS model power consumption prediction. In the main evaluation experiment, the model showed an average power estimation error of -0.66%, with the vast majority of prediction errors not exceeding an absolute value of 3%. These results are further supported by the experiments we conducted to show the tradeoff between schedule length and steady-state temperature, as well as by the preliminary results of additional experiments that are taking place at the time of writing. All of the described evaluation was performed on real hardware.

With all of this in mind, we consider the goals of this thesis to be fulfilled.

The next step in the research is to test (and possibly improve, depending on the results) the PEWMS model on a more varied task set, such as memory-intensive workloads or workloads utilizing the GPU. We focused on working with primarily CPU-intensive benchmarks, and we only slightly touched on stressing the memory with the membench benchmarks. During the evaluation, we encountered an outlier case in which the PEWMS model made an estimation error of -7.69% in an instance that included the membench tasks. This suggests that there may be circumstances under which the PEWMS model does not perform as well as we would expect it to.

References

[1] MCIMX8QM-CPU: i.MX 8QuadMax Multisensory Enablement Kit (MEK). https://www.nxp.com/design/development-boards/i-mx-evaluation-and-development-boards/i-mx-8quadmax-multisensory-enablement-kit-mek:MCIMX8QM-CPU. [Online] Accessed: 2021-04-20.

[2] INA219 data sheet, product information and support. https://www.ti.com/product/INA219. [Online] Accessed: 2021-05-07.

[3] ARINC Specification. 653-2: Avionics application software standard interface: Part 1-required services. Technical report, Technical report, Avionics Electronic Engineering Committee (ARINC), 2006.

[4] Hafiz Fahad Sheikh, Ishfaq Ahmad, Zhe Wang, and Sanjay Ranka. An overview and classification of thermal-aware scheduling techniques for multi-core processing systems. *Sustainable Computing: Informatics and Systems*, 2(3):151–169, 2012.

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

[6] Jian-Jia Chen, Andreas Schranzhofer, and Lothar Thiele. Energy minimization for periodic real-time tasks on heterogeneous processing units. In *2009 IEEE International Symposium on Parallel Distributed Processing*, pages 1–12, 2009.

[7] Yang Qin, Gang Zeng, Ryo Kurachi, Yutaka Matsubara, and Hiroaki Takada. Energy-aware task allocation for heterogeneous multiprocessor systems by using integer linear programming. *Journal of Information Processing*, 27:136–148, 2019.

[8] Junlong Zhou, Tongquan Wei, Mingsong Chen, Jianming Yan, Xiaobo Sharon Hu, and Yue Ma. Thermal-aware task scheduling for energy minimization in heterogeneous real-time mpsoc systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(8):1269–1282, 2016.

[9] Chin-Fu Kuo and Yung-Feng Lu. Task assignment with energy efficiency considerations for non-dvs heterogeneous multiprocessor systems. *SIGAPP Appl. Comput. Rev.*, 14(4):8–18, January 2015.

[10] Marco ET Gerards, Johann L Hurink, and Philip KF Hölzenspies. A survey of offline algorithms for energy minimization under deadline constraints. *Journal of Scheduling*, 19(1):3–19, 2016.

[11] Pengcheng Huang, Georgia Giannopoulou, Rehan Ahmed, Davide B Bartolini, and Lothar Thiele. An isolation scheduling model for multicores. In *2015 IEEE Real-Time Systems Symposium*, pages 141–152. IEEE, 2015.

[12] Peter Brucker, Andrei Gladky, Han Hoogeveen, Mikhail Y Kovalyov, Chris N Potts, Thomas Tautenhahn, and Steef L Van De Velde. Scheduling a batching machine. *Journal of scheduling*, 1(1):31–54, 1998.

[13] CTU-IIG/demos-sched: Scheduler for simulation of avionics multi-core workloads on Linux. `https://github.com/CTU-IIG/demos-sched`. [Online] Accessed: 2021-05-05.

[14] Basics of the Unix Philosophy. `https://homepage.cs.uri.edu/~thenry/resources/unix_art/ch01s06.html`. [Online] Accessed: 2021-04-18.

[15] nlohmann/json: JSON for Modern C++. `https://github.com/nlohmann/json`. [Online] Accessed: 2021-04-18.

[16] Gurobi - The fastest solver. `https://www.gurobi.com/`. [Online] Accessed: 2021-04-18.

[17] The CImg Library - C++ Template Image Processing Toolkit. `https://cimg.eu/`. [Online] Accessed: 2021-04-18.

[18] lvandeve/lodepng: PNG encoder and decoder in C and C++. `https://github.com/nlohmann/json`. [Online] Accessed: 2021-04-18.

[19] Heiko Falk, Sebastian Altmeyer, Peter Hellinckx, Björn Lisper, Wolfgang Puffitsch, Christine Rochange, Martin Schoeberl, Rasmus Bo Sørensen, Peter Wägemann, and Simon Wegener. Taclebench: A benchmark collection to support worst-case execution time research. In *16th International Workshop on Worst-Case Execution Time Analysis*, 2016.

[20] Tiny renderer or how OpenGL works: software rendering in 500 lines of code. `https://github.com/ssloy/tinyrenderer`. [Online] Accessed: 2021-05-03.

[21] Michal Sojka, Ondřej Benedikt, Zdeněk Hanzálek, and Pavel Zaykov. Testbed for thermal and performance analysis in mpsoc systems. In *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*, pages 683–692. IEEE, 2020.

[22] AutoBench – Automotive Industrial Benchmarking - EEMBC Embedded Microprocessor Benchmark Consortium. https://www.eembc.org/autobench/. [Online] Accessed: 2021-05-13.

Appendix A

List of abbreviations

- **CPU:** Central Processing Unit
- **DPM:** Dynamic Power Management
- **DVFS:** Dynamic Voltage Frequency Scaling
- **GPU:** Graphics Processing Unit
- **ILP:** Integer Linear Programming
- **LTF:** Longest Tasks First
- **MEK:** Multisensory Enablement Kit
- **MPSoC:** Multiprocessor System on a Chip
- **PEWMS:** Power Estimation With Max Static power term

Appendix B

Contents of the attached CD

- `tools` – Directory containing the source code of the developed software. For instructions on compiling and using it, please see the `README.md` file inside this directory.
- `test_instances` – Directory containing the JSON representation used by our tools of the evaluated schedules.
- `experiment_results` – Directory containing an unprocessed output from the conducted experiments.
- `thesis/latex` – Directory containing the LaTeX source code of this thesis.
- `thesis/thesis.pdf` – This thesis in PDF format.

Appendix C

Comparison of power estimation functions

Here we show the power estimation results for alternative power estimation functions that showed promise during the early stages of the research and compare them to the PEWMS function, which ultimately had the best results. The comparison is performed on the schedules for the six instances from Chapter 7.

To recapitulate, the PEWMS power estimation function is:

$$P_{MF} = \frac{1}{h} \sum_{W_j \in \mathcal{W}} E_j \tag{C.1}$$

$$E_j = l_j \cdot P_{idle} + \sum_{T_i \in \mathcal{T}: a_w(T_i) = W_j} \sum_{R_k \in \mathcal{R}: a_r(T_i) = R_k} p_{i,k} \cdot a_{i,k} + l_j \cdot \max_{\substack{T_i \in \mathcal{T}: a_w(T_i) = W_j \\ R_k \in \mathcal{R}: a_r(T_i) = R_k}} b_{i,k} \tag{C.2}$$

The other functions we present here are the No Static Power (NOSTAP), which does not work with the static power consumption coefficients:

$$P_{MF} = \frac{1}{h} \sum_{W_j \in \mathcal{W}} \left(l_j \cdot P_{idle} + \sum_{T_i \in \mathcal{T}: a_w(T_i) = W_j} \sum_{R_k \in \mathcal{R}: a_r(T_i) = R_k} p_{i,k} \cdot a_{i,k} \right) \tag{C.3}$$

And the Proportional Estimation of Static Power (PESP) function, which considers the static power consumption coefficient to add up with the dynamic power consumption proportionally to the resource capacity:

$$P_{MF} = \frac{1}{h} \sum_{W_j \in \mathcal{W}} \left(l_j \cdot P_{idle} + \sum_{T_i \in \mathcal{T}: a_w(T_i) = W_j} \sum_{R_k \in \mathcal{R}: a_r(T_i) = R_k} p_{i,k} \cdot \left(a_{i,k} + \frac{b_{i,k}}{c_k} \right) \right) \tag{C.4}$$

Because of the amount of data, the comparison of the power estimation functions is shown in two separate tables: Table C.1 and Table C.2.

The NOSTAP function exhibits the greatest estimation error, on average 6.35%, which can be attributed to the function not accounting for the static power consumption. The PESP function exhibits an average estimation error of 1.49%. Compared to the PEWMS function, which exhibits an average

estimation error of -0.66%, the other two functions tend to underestimate the power consumption. We consider the slight overestimation that the PEWMS function manifests to be better than an underestimation for our purposes.

Power estimation error ϵ [%]

	global-ILP			minutil+LTF			Ref. assgmnt. heuristic+LTF		
Instance	PEWMS	NOSTAP	PESP	PEWMS	NOSTAP	PESP	PEWMS	NOSTAP	PESP
#1	-0.09	4.16	-0.36	-0.53	4.14	0.81	-0.16	5.79	0.91
#2	0.15	4.49	0.14	0.94	5.09	2.02	-0.33	7.05	2.65
#3	-1.70	2.31	-1.17	-0.32	2.67	0.08	-2.66	0.99	-2.63
#4	1.50	5.80	1.80	0.60	6.86	2.42	0.95	6.86	2.23
#5	-0.42	5.55	-1.53	-0.64	7.42	1.49	-7.69	10.20	1.25
#6	-0.24	5.69	-0.47	0.59	6.52	1.86	-2.88	7.33	0.92

Table C.1: Comparison of power estimation errors of other functions (part 1)

Power estimation error ϵ [%]

	RA+LTF1			RA+LTF2			RA+LTF3		
Instance	PEWMS	NOSTAP	PESP	PEWMS	NOSTAP	PESP	PEWMS	NOSTAP	PESP
#1	-0.13	6.89	2.62	-1.69	6.82	2.41	-1.60	5.04	1.06
#2	0.06	7.56	3.41	-0.26	5.65	1.72	-2.73	5.05	0.94
#3	-2.23	1.13	-2.07	2.72	6.08	2.47	-0.56	3.49	-0.18
#4	-0.66	7.87	2.63	1.24	6.66	2.28	-1.24	5.87	0.60
#5	0.90	11.35	4.71	0.64	11.66	4.81	-0.51	10.22	3.87
#6	-2.87	7.58	1.81	-1.04	10.49	4.16	-1.05	10.14	3.83

Table C.2: Comparison of power estimation errors of other functions (part 2)

Appendix D

Visualization of the evaluated schedules

Here we provide a visualization of all the evaluated schedules from Chapter 7. Table D.1 lists the major frame lengths of the individual instances.

Instance	Figure	h [ms]
#1	Figure D.1	1300
#2	Figure D.2	1330
#3	Figure D.3	1000
#4	Figure D.4	1150
#5	Figure D.5	1170
#6	Figure D.6	1290

Table D.1: Major frame lengths of the evaluated instances

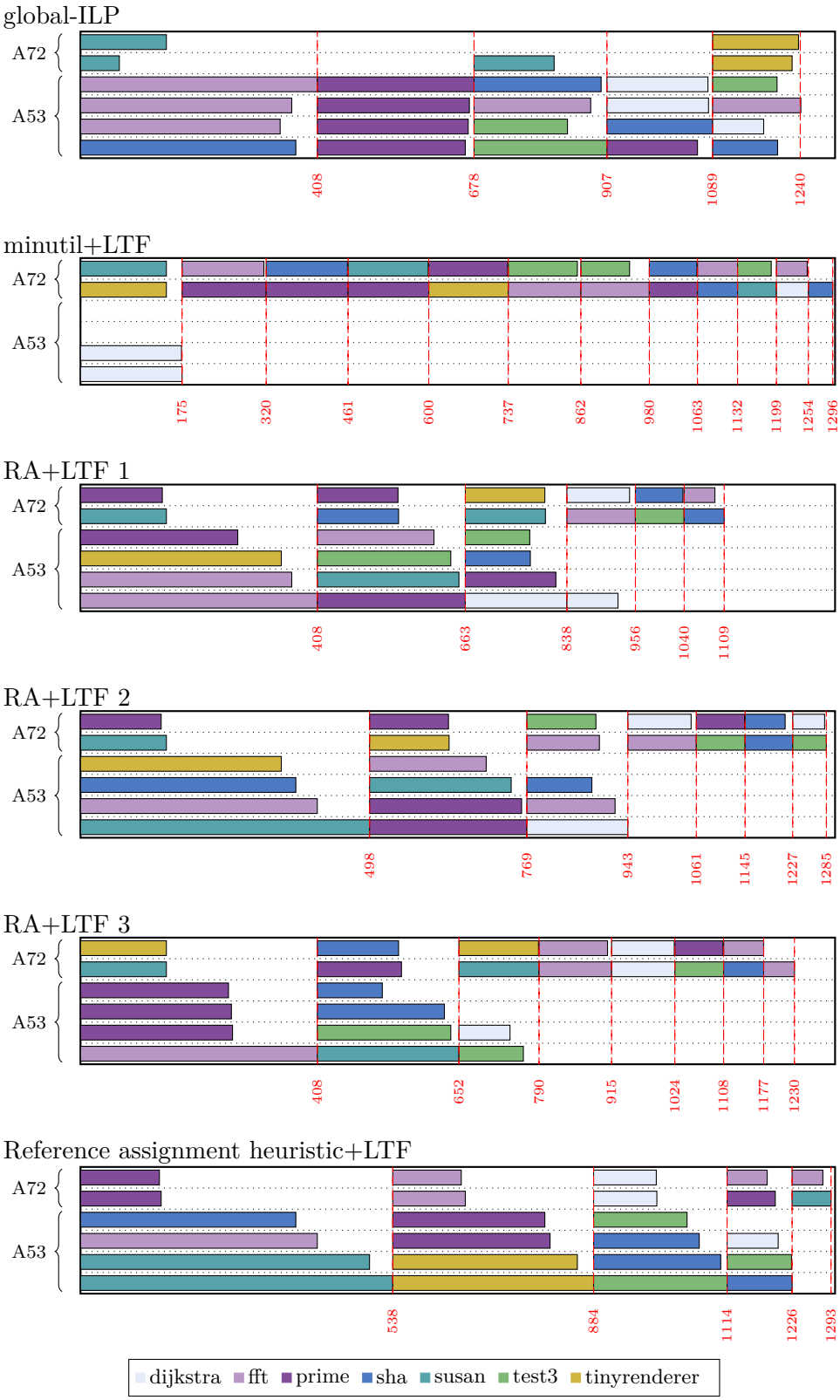


Figure D.1: Schedules for Instance #1

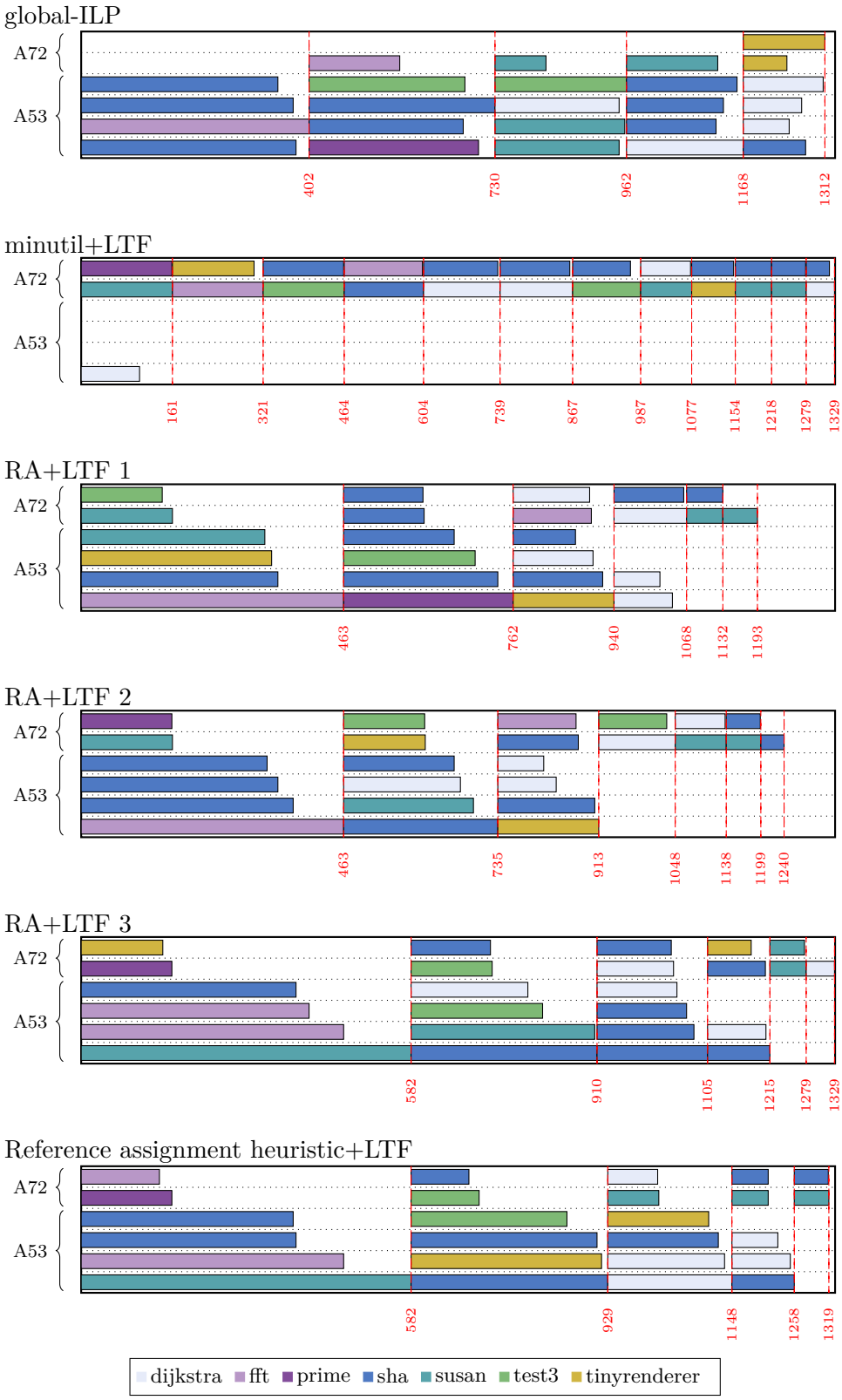


Figure D.2: Schedules for Instance #2

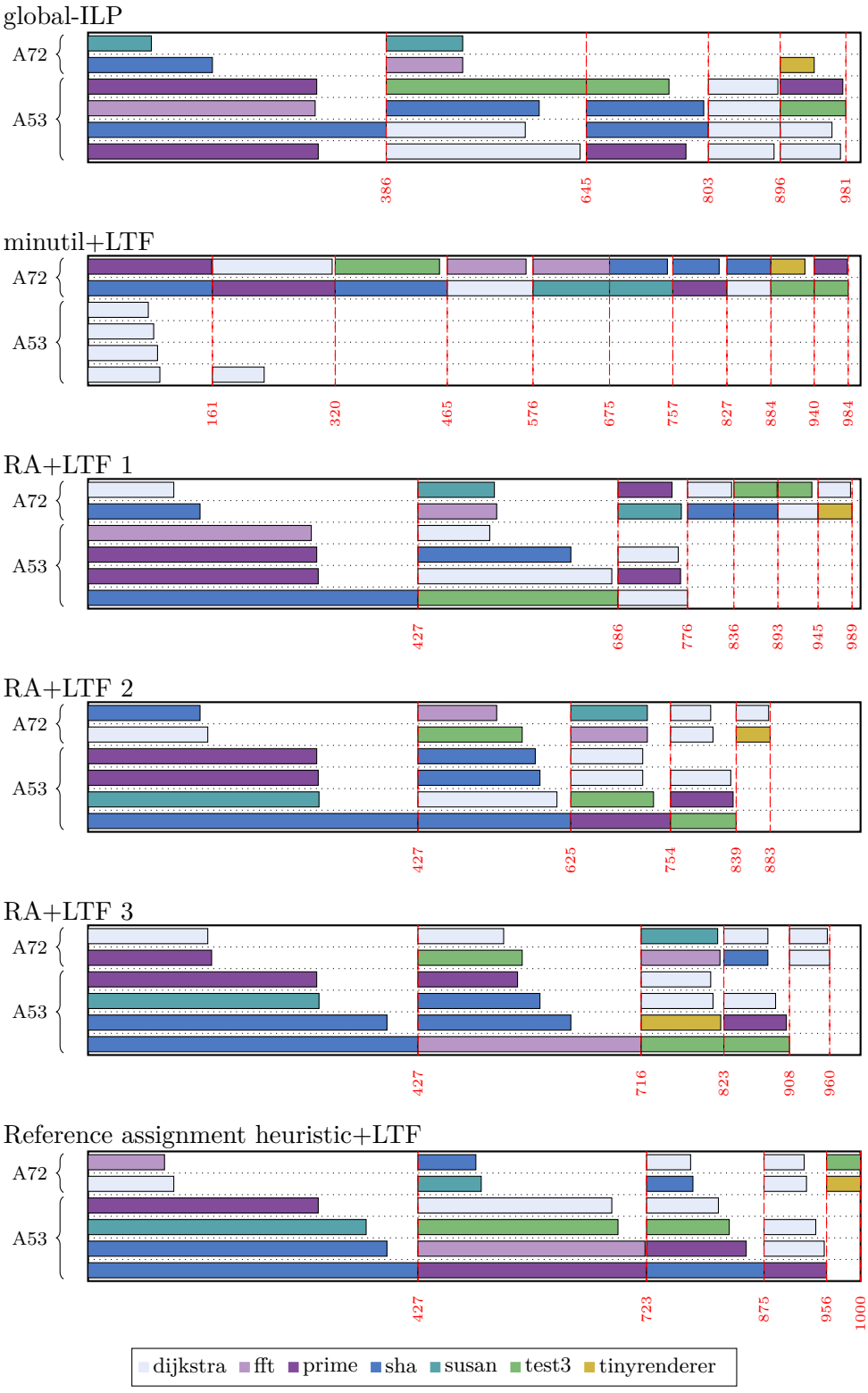


Figure D.3: Schedules for Instance #3

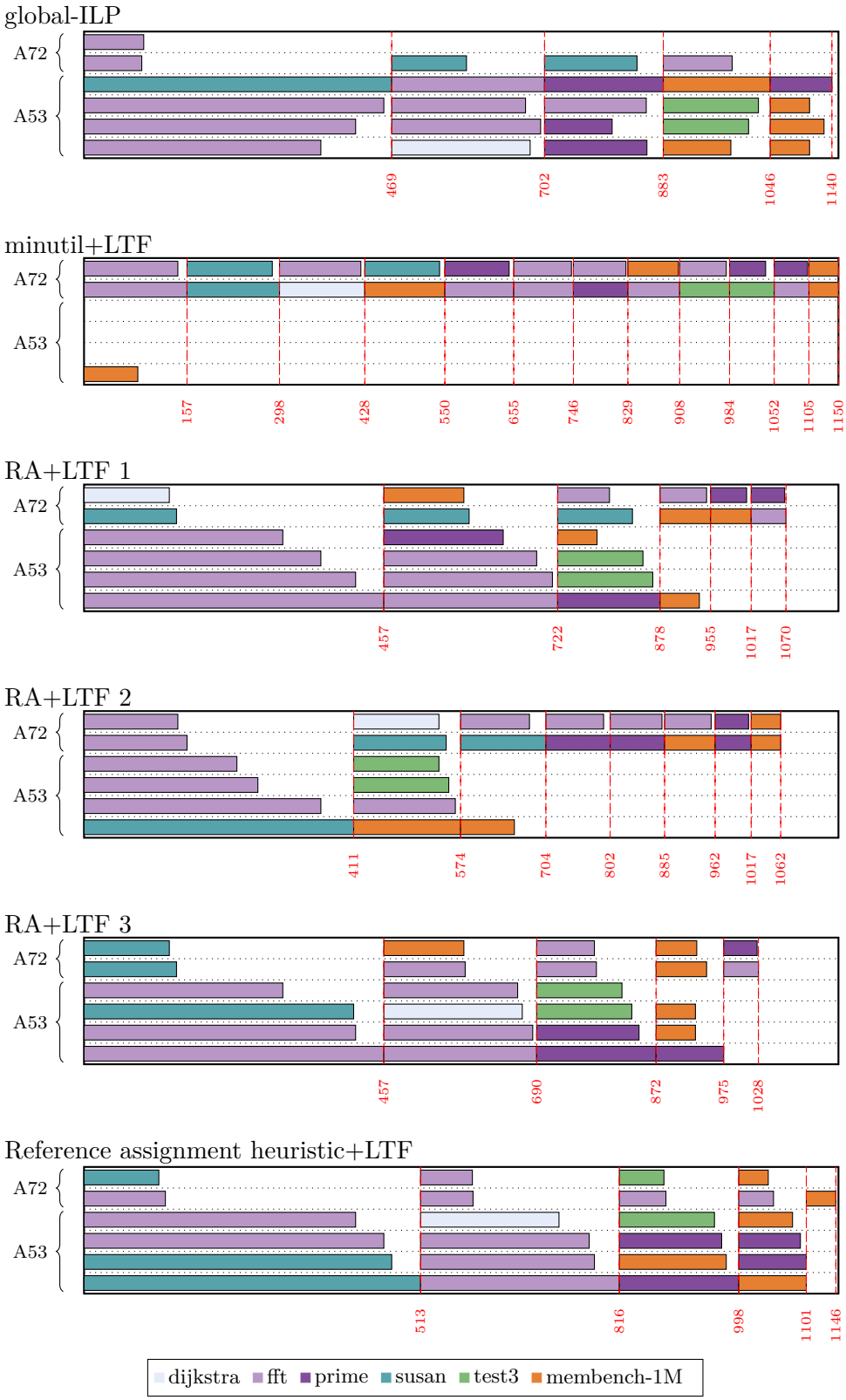


Figure D.4: Schedules for Instance #4

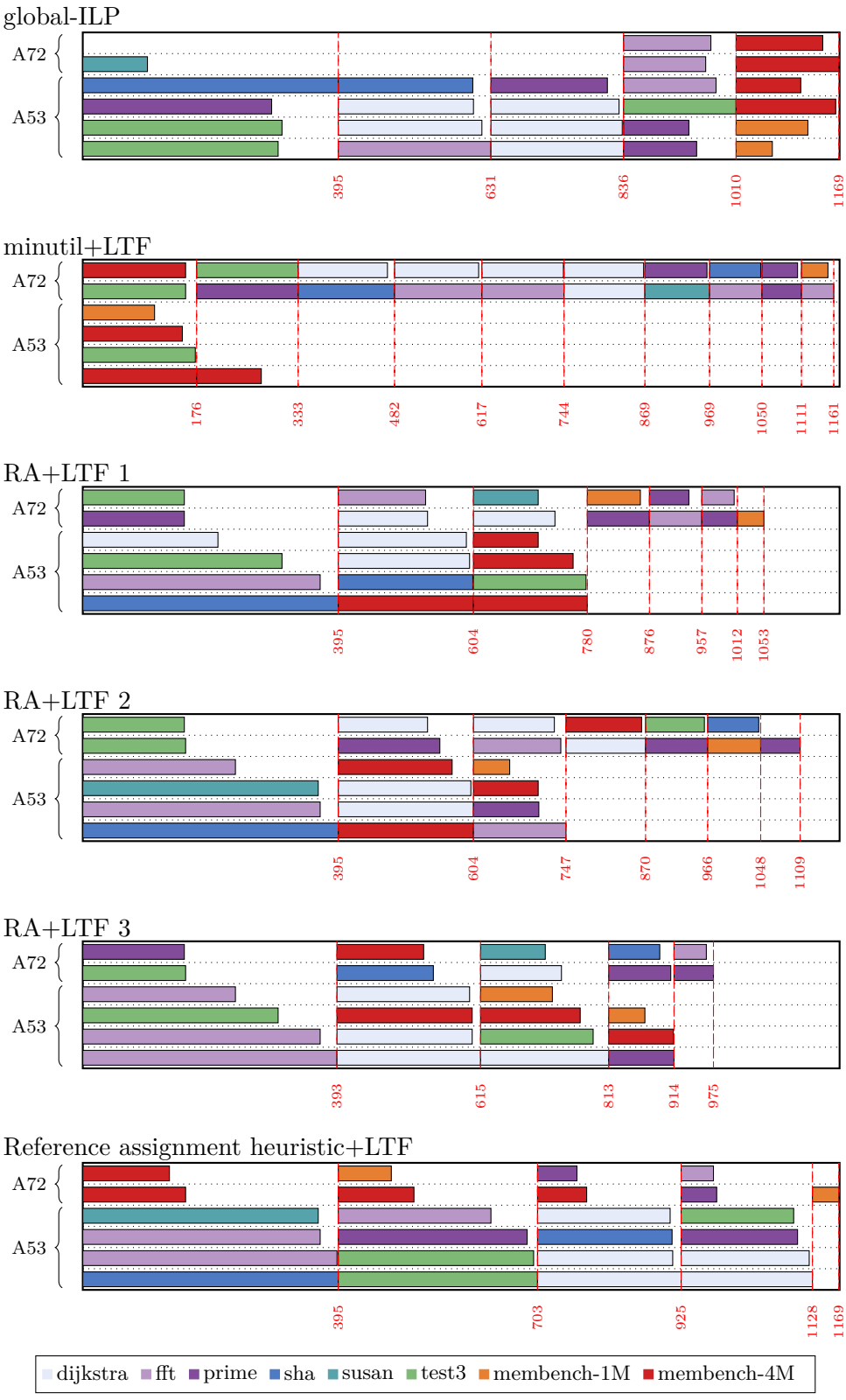


Figure D.5: Schedules for Instance #5

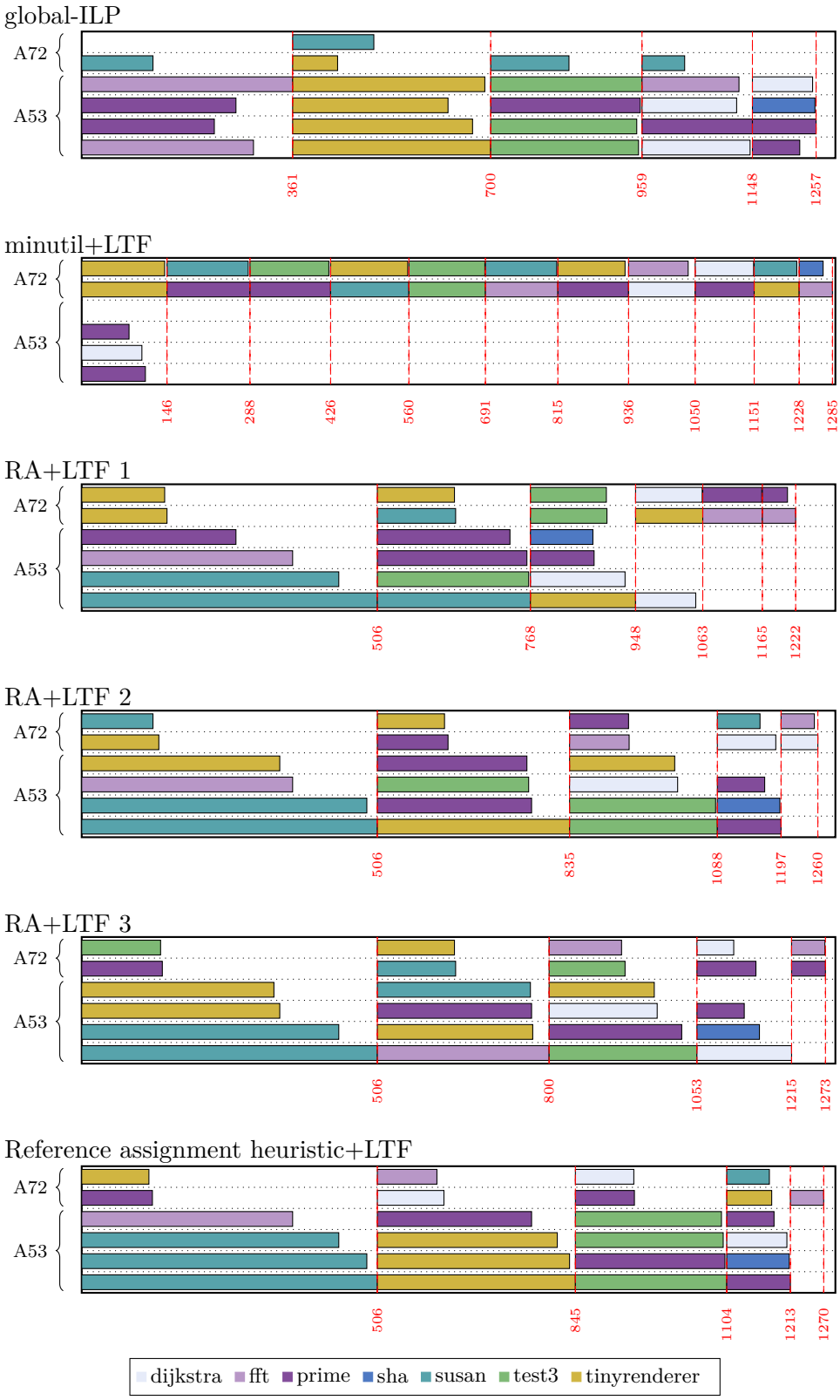


Figure D.6: Schedules for Instance #6

Appendix E

Measured and estimated average power consumption values

Table E.1 and Table E.2 show the measured (averaged across all three runs) and estimated average power consumption of the schedules evaluated in Chapter 7.

	Average power consumption [W]					
	global-ILP		minutil+LTF		Ref. assgmnt. heuristic+LTF	
Instance	Measured	Estimated	Measured	Estimated	Measured	Estimated
#1	7.313 ± 0.016	7.320	8.276 ± 0.009	8.321	7.638 ± 0.027	7.650
#2	7.410 ± 0.014	7.398	8.476 ± 0.003	8.397	7.782 ± 0.024	7.808
#3	7.232 ± 0.027	7.357	8.221 ± 0.031	8.248	7.239 ± 0.100	7.437
#4	7.435 ± 0.044	7.324	8.333 ± 0.026	8.284	7.549 ± 0.039	7.478
#5	7.396 ± 0.023	7.427	8.474 ± 0.031	8.529	7.659 ± 0.045	8.297
#6	7.410 ± 0.009	7.428	8.493 ± 0.037	8.443	7.698 ± 0.012	7.927

Table E.1: Measured and estimated average power consumptions of the evaluated schedules (part 1)

	Average power consumption [W]					
	RA+LTF 1		RA+LTF 2		RA+LTF 3	
Instance	Measured	Estimated	Measured	Estimated	Measured	Estimated
#1	7.918 ± 0.020	7.928	7.960 ± 0.026	8.097	7.916 ± 0.012	8.045
#2	7.992 ± 0.015	7.988	7.932 ± 0.007	7.952	7.769 ± 0.024	7.987
#3	7.483 ± 0.130	7.654	7.767 ± 0.043	7.561	7.605 ± 0.206	7.648
#4	7.753 ± 0.038	7.804	7.968 ± 0.038	7.870	7.533 ± 0.024	7.627
#5	8.236 ± 0.034	8.163	8.380 ± 0.043	8.326	8.098 ± 0.030	8.140
#6	7.962 ± 0.006	8.197	7.990 ± 0.034	8.074	8.031 ± 0.008	8.116

Table E.2: Measured and estimated average power consumptions of the evaluated schedules (part 2)

Appendix F

Preliminary results of additional experiments

The problem instances of the additional experiments we present here consist mainly of benchmarks from the well-established EEMBC Autobench benchmark suite [22] often used in the automotive industry. The suite is comprised of 12 different benchmarks, each with two different workload sizes: 4K and 4M. The benchmarks include signal processing algorithms, for instance, finite impulse response filter and inverse discrete cosine transform, automotive algorithms like road speed calculation, as well as generic workloads such as matrix arithmetic.

We evaluated schedules for instances with only a single workload size, instances with combined workload sizes, as well as instances with combinations of benchmarks from the Autobench and the Taclebench suite. The thermal results are shown in Table F.1. Table F.2 contains the evaluation of the power estimation error for schedules of these instances. So far, only a single measurement for each schedule was performed. Still, we believe the results can be considered representative since they conform to the observations made on the Taclebench instances in Chapter 7.

The steady-state temperature of the global-ILP schedules was on average better by 13.23% than minutil+LTF schedules, by 6.57% better than RA+LTF schedules, and by 3.23% better than schedules based on the reference assignment heuristic. Interestingly, there were a few cases in which the reference assignment heuristic schedules were slightly better (by less than 1%) than the global-ILP schedules. This could have been caused by an inaccuracy in the measurement (possibly because of an ambient temperature change during the experiment), considering the temperature difference is almost insignificant.

The average power estimation error of the PEWMS model across all the evaluated schedules was -0.78%.

Instance	T_{rel} [°C]			
	global-ILP	minutil+LTF	RA+LTF	Ref. assgmnt. heuristic+LTF
Autobench 4K #1	28.09	33.15	30.04	29.36
Autobench 4K #2	29.08	32.94	30.72	29.03
Autobench 4K #3	29.37	33.07	31.82	29.14
Autobench 4K + Taclebench #1	29.00	33.28	31.37	29.87
Autobench 4K + Taclebench #2	28.94	32.84	31.54	29.61
Autobench 4K + Taclebench #3	30.44	35.59	32.20	32.31
Autobench 4M #1	35.14	39.17	37.47	35.65
Autobench 4M #2	35.95	39.89	37.72	35.86
Autobench 4M #3	36.45	41.37	37.00	37.07
Autobench 4M + Taclebench #1	32.59	38.99	35.28	34.76
Autobench 4M + Taclebench #2	35.17	40.91	37.73	36.27
Autobench 4M + Taclebench #3	34.03	38.80	35.45	34.85
Autobench 4M + Autobench 4K #1	32.54	38.75	35.61	34.76
Autobench 4M + Autobench 4K #2	33.54	37.78	35.89	34.48
Autobench 4M + Autobench 4K #3	33.75	39.17	36.90	35.70
Autobench 4M + Autobench 4K + Taclebench #1	30.55	35.73	32.98	32.42
Autobench 4M + Autobench 4K + Taclebench #2	31.95	38.75	34.60	33.90
Autobench 4M + Autobench 4K + Taclebench #3	33.07	37.90	35.92	34.15

Table F.1: Relative steady-state temperatures of the additional experiment schedules

Instance	Power estimation error ϵ [%]			
	global-ILP	minutil+LTF	RA+LTF	Ref. assgmnt. heuristic+LTF
Autobench 4K #1	-1.63	0.48	-0.36	-0.87
Autobench 4K #2	-0.96	-0.36	0.44	-1.34
Autobench 4K #3	-0.81	-0.17	0.51	-1.21
Autobench 4K + Taclebench #1	-0.35	1.25	0.52	0.15
Autobench 4K + Taclebench #2	0.79	1.22	1.59	-1.57
Autobench 4K + Taclebench #3	1.46	2.63	1.32	1.11
Autobench 4M #1	1.00	-0.33	-2.27	-4.21
Autobench 4M #2	0.38	-0.11	-3.05	-4.98
Autobench 4M #3	1.13	-0.29	-2.85	-3.61
Autobench 4M + Taclebench #1	0.52	1.74	-1.05	-1.80
Autobench 4M + Taclebench #2	0.04	0.69	-1.23	-2.32
Autobench 4M + Taclebench #3	1.91	0.01	-3.95	-3.72
Autobench 4M + Autobench 4K #1	-0.71	0.59	-0.42	-0.61
Autobench 4M + Autobench 4K #2	-0.96	0.34	-1.53	-1.52
Autobench 4M + Autobench 4K #3	-0.84	-1.56	-2.54	-4.68
Autobench 4M + Autobench 4K + Taclebench #1	-0.28	1.07	-0.10	-2.49
Autobench 4M + Autobench 4K + Taclebench #2	-1.13	1.16	-3.38	-4.62
Autobench 4M + Autobench 4K + Taclebench #3	0.52	-0.32	-3.74	-3.60

Table F.2: Power estimation errors of the PEWMS model for the additional experiment schedules

Appendix G

Scalability of the global-ILP model

To evaluate the scalability of the global-ILP model, we measured the time needed to find an optimal solution for $|\mathcal{T}| = \{20, 22, 24, 26, 28, 30\}$ on five different instances. The time limit of 10 minutes was not enough to find the optimal solution of the global-ILP model for instances with 32 and more tasks. For comparison, we also measured the time needed for the reference assignment heuristic to find a solution. The instances were randomly generated, with the same execution time and major frame length parameters as described in Section 7.1.

The evaluation was performed on a system with four Intel Xeon 4110 CPUs running at 2.1 GHz and 192 GB of RAM. The ILP solver used was Gurobi, version 9.0. The average times needed for finding a solution are listed in Table G.1 and shown in a graph in Figure G.1. The times strongly depend on the tightness of a particular instance, hence the considerable variance in results.

	Solve time [s]	
$\|\mathcal{T}\|$	global-ILP	Reference assignment heuristic
20	1.9 ± 0.4	1.0 ± 0.1
22	3.7 ± 2.0	2.2 ± 0.7
24	7.9 ± 3.1	3.0 ± 0.4
26	33.3 ± 16.3	4.1 ± 0.8
28	87.9 ± 22.2	9.0 ± 1.1
30	100.1 ± 44.9	13.2 ± 2.4

Table G.1: Average time needed to find an optimal solution

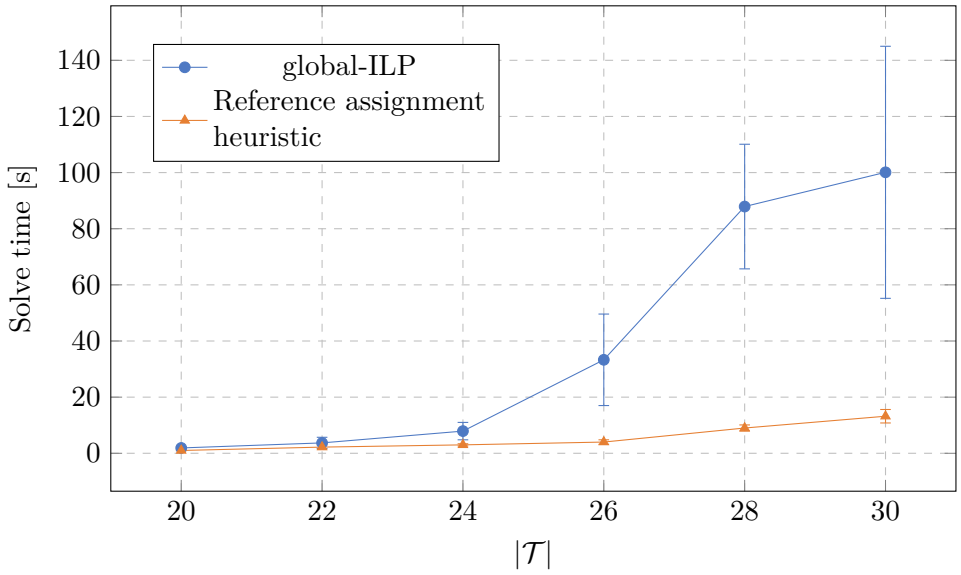


Figure G.1: Average time needed to find an optimal solution