Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Master Thesis
at the Department of Information Technology
and Electrical Engineering

Thermal Simulation and Analysis Methods for Many-Core Platforms

AS 2010

Lars Schor

Advisors:	Dr. Iuliana Bacivarov
	Dr. Hoeseok Yang
Professor:	Prof. Dr. Lothar Thiele

Zurich
30th January 2011

Abstract

The integration of multiple processors on a single chip and three-dimensional stacking are both considered as future technologies in microprocessor design to achieve unprecedented performance. However, the integration of multiple processors on a single chip increases the power density, and three-dimensional stacking makes cooling almost impossible. This leads to various drawbacks with regard to the reliability of a system, and consequently, it becomes infeasible to guarantee real-time constraints. Therefore, new system-level methodologies are required that guarantee both thermal constraints and real-time deadlines at the design time.

This master thesis introduces a novel approach to analyze the transient thermal behavior of a many-core system in an early design stage and proposes SLTE, a compositional thermal evaluation model. A low-level thermal evaluation tool chain is designed to automatically calibrate the proposed model with the required thermal and timing parameters. A prototype implementation of SLTE is implemented in DOL and is used to evaluate the model in three case studies. In comparison with the low-level tool chain, SLTE offers significant speed-ups in the order of three magnitudes while preserving a high accuracy, that is, the difference between the maximum temperatures is less than one percent.

Calculating an upper bound of the maximum temperature is crucial for embedded real-time systems. This master thesis proposes an analytic framework to calculate the maximum temperature of a many-core system under all possible scenarios of task executions. All event-streams are modeled as arrival curves and the only requirement towards the processing components is that the real-time scheduling algorithms are work-conserving. As a special case, periodic streams with bounded jitter and bursts are considered to evaluate the framework in several case studies. The achieved results highlight that the impact of the proposed framework on the design process of embedded real-time systems is fundamental.

Acknowledgements

First of all I would like to express my sincere gratitude to Prof. Dr. Lothar Thiele for giving me the opportunity to write this master thesis in his research group. All the fruitful discussions about my problems and the promising approaches on how to solve them had notably influence on the outcome of this thesis.

I would also like to thank my advisors Dr. Iuliana Bacivarov and Dr. Hoeseok Yang for their constant support during this master thesis. They always asked the right questions and helped me to solve the difficult problems of this thesis. Without their assistance, this work would never have been possible. In particular, I would like to thank them for giving me the opportunity to take part at the thermal meeting at EPFL, which introduced me to the research field of thermal analysis.

Furthermore, I would like to express my thankfulness to Dr. Wolfgang Haid for introducing me to the challenges of distributed systems and sharing his joy for multi-processors systems with me.

Finally, my warmest thanks go to my family and my girlfriend for supporting and motivating me during my studies.

Contents

List of Figures

List of Tables

List of Algorithms

Listings

1 Introduction

1.1 Motivation

Current trends in microprocessor design include the integration of multiple processors on a single chip. The widely used Cell Broadband Engine [1] from STI, that was first delivered in 2006, integrates eight data processing engines on a single chip. The newly announced Intel Single-chip Cloud Computer (SCC) [2] already integrates 48 cores to stay abreast of the ever increasing demand of computation performance. However, the obtained increase in performance imposes a major increase in power density, which in turn may create hot spots on the die.

The continuous increase in power density leads to various drawbacks with regard to the reliability of a system. The excess of a threshold temperature could lead to a reduction of the system performance, a reduced lifetime, or even a damage of the physical system. The cooling costs of such systems highly increase with the number of processing engines and sufficient cooling becomes even infeasible in various embedded systems like cell phones.

To tackle the new challenges of many-core systems, three-dimensional stacking is even considered as future technology to achieve unprecedented performance. However, the placement of computational units on top of each other continuously increases the power density and complicates the task of cooling the system.

Reactive thermal management techniques that are widely used at the hardware-level of high-performance systems can lead to a major loss in performance. Like hardware-level techniques, the use of Operating System (OS) schedulers that take into account thermal management, can lead to renege on real-time constraints, and complicates the predication of the system behavior. Therefore, new system-level methodologies that guarantee both thermal constraints and real-time deadlines at design time are required. Analyzing thermal issues only in a late stage of the design cycle can have adverse effects including increased hardware costs or an abortion of the project. Therefore, thermal analysis methods are required that address the problem of evaluating the temperature already in an early design stage when the system is often only described by its software synthesis specifications.

This master thesis addresses these challenges and proposes solutions to estimate the temperature already in a very early design stage. Methods to both estimate the transient temperature behavior and calculate the worst-case peak temperature of a many-core system are proposed. Finally, the suggested methods are implemented in Distributed Operation Layer (DOL) [3, 4] or Modular Performance Analysis (MPA) [5].

1.2 Context

This master thesis is carried out in the framework of the European project PRO3D to which the ETH Zurich is contributing in terms of performance analysis and mapping optimization. In the following, the PRO3D project, DOL, and the main techniques for mapping optimization are briefly introduced.

1.2.1 PRO3D

The aim of the European project PRO3D is the devolvement of a system design methodology for future embedded computing. On the one hand, from the hardware perspective, its outputs are new three-dimensional architecture concepts for many-core platforms. On the other hand, from the software perspective, the goal is the development of a system software flow that guarantees the correct operation of hardware and software. Therefore, new formal methods are required for evaluating performance and thermal impacts already in an early design stage.

The target platform of the project is a novel industrial many-core System On Chip (SoC), namely the *Platform 2012* [6] provided by STMicroelectronics. Besides a specific version of the Multiprocessor ARM (MPARM) virtual platform for three-dimensional architectures, the *Platform 2012* will be used as case study for three-dimensional developments.

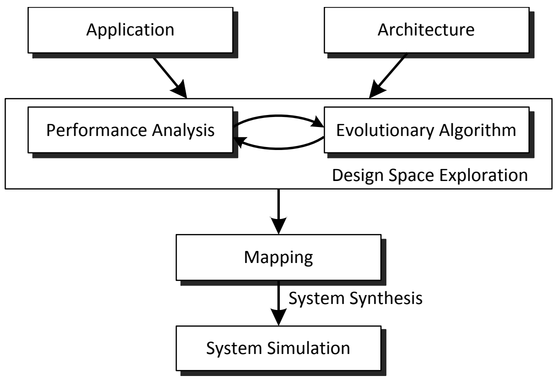


Figure 1.1: The Y-Chart design methodology as it is used in DOL.

1.2.2 Distributed Operation Layer (DOL)

DOL [3, 4] is a platform independent programming framework for Multiprocessor System-on-Chips (MPSoCs), which implements parallel real-time applications onto many-core architectures. Originally developed during the project SHAPES [7], DOL is currently extended within the European projects EURETILE[1] and PRO3D at the ETH Zurich. The thermal evaluation methods developed in this thesis will be used to extend DOL with the functionality to analyze thermal-aware constraints.

Following the Y-Chart [8] design methodology outlined in Fig. 1.1, DOL consists of an application programming interface for multi-processor systems, a functional simulation and a mapping optimization. The DOL programming model uses the Kahn Process Network (KPN) [9] semantic as model of computation. The behavior of the application is defined in C/C++ and an XML representation has been selected to syntactically describe the coordination of the process network so that the application can be defined in an architecture-independent way. Recently, DOL has already been coupled with MPARM [10], the target platform for this thesis.

1.2.3 Design Space Exploration

The design space exploration describes the process of exploring various design alternatives of a system to finally decide, which one to implement [11]. In this thesis, design space exploration refers to the task of exploring the optimal mapping of a multi-processor streaming application onto a distributed memory architecture in a time and thermal efficient manner. The corresponding flow-chart is outlined in Fig. 1.2.

[1]`http://euretile.roma1.infn.it`

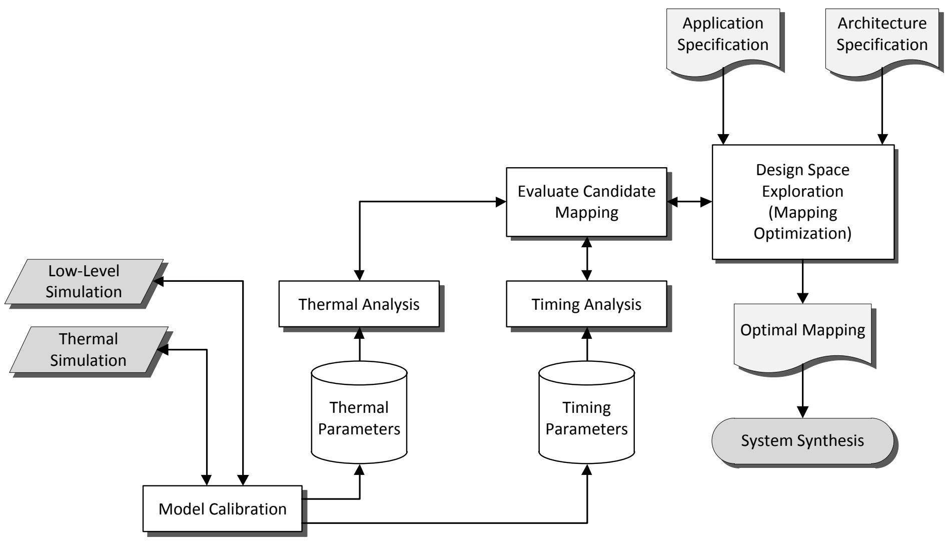


Figure 1.2: Flow-chart of the design space exploration of a multi-processor streaming application that is mapped onto a distributed architecture.

The design flow can be summarized as follows: Prior to the design space exploration, the *Model Calibration* is performed, where timing and thermal parameters are extracted from low-level and thermal simulation, respectively. During the actual design space exploration, the analysis models fall back on these parameters to model the actual timing and thermal behavior of the system. Due to their well-known intractability of the problem, evolutionary algorithm based approaches are often used during the design space exploration to explore the design space [3, 12]. The performance and thermal characteristics of every *Candidate Mapping* is analyzed during the *Timing* and *Thermal Analysis*, respectively. Finally, the output of the design space exploration is the optimal mapping of a given multi-processor application onto a distributed memory architecture.

In PRO3D, DOL will be used to calculate the mapping in a thermal and timing efficient manner. While DOL already supports mapping optimization with respect to timing constraints, in this thesis, DOL will be extended with the capability to collect thermal-aware metrics in order to support mapping optimizations with respect to thermal-aware constraints.

1.3 Contributions

This master thesis addresses the task of thermal evaluation in the design optimization process of a multi-processor streaming application. This mainly includes the development of methods to quickly evaluate thousands of candidate mappings during design space exploration in terms of thermal aspects.

Evaluation methods for both the transient temperature evolution and worst-case peak temperature of a many-core system that is only described by its software synthesis specifications, are proposed. In the following, a list of contributions of this thesis is provided.

Thermal Evaluation Tool Chain. Multiple steps are required to simulate the temperature of an application described by its software synthesis specifications. We discuss an approach that concatenates three different tools, namely a software synthesis tool, a low-level virtual platform and a thermal analysis model to calculate the transient thermal behavior of an application. The proposed solution is integrated in the DOL design flow using the MPARM virtual platform [13] and HotSpot [14]. Besides the concatenation of the three tools into one chain, the tools have been extended in this thesis with the following functionalities. The DOL synthesis tool is extended to automatically generate the architecture information, namely the floor-plan and the thermal parameters. Two major upgrades are required for the MPARM virtual platform:

1. The calculation of the transient energy and power consumption from the already existing functionality to compute the overall energy and the average power consumption.
2. The extension of the power models with the ability to calculate the static power consumption and to distinguish between the static and dynamic power consumption.

Finally, a software wrapper for HotSpot has been developed that is able to handle the power consumption as provided by the MPARM virtual platform and to reevaluate the static power consumption as a quadratic function of the current temperature.

Automated Calibration of Thermal Evaluation Models. Compositional thermal analysis models have many advantages over low-level based evaluation methods during design space exploration, but they have the disadvantage of requiring timing and thermal parameters to model the behavior of a multi-processor streaming application. In this thesis, we propose methods for the automated calibration of abstract thermal analysis models during which all the required data is collected, and after post-processing, added to the system specifications. The proposed solution is implemented in DOL for the automated extraction of the timing and thermal parameters of a multi-processor streaming application targeting the MPARM virtual platform.

Transient Temperature Emulation at System-Level. The first main contribution of this thesis is a compositional thermal evaluation model to estimate the transient temperature evolution of a multi-processor application. We discuss the computation of the transient power consumption using a trace-based simulation framework and the calculation of the temporal temperature behavior by means of a precomputed thermal analysis method. We implement the proposed model in DOL targeting the MPARM virtual platform and show that speed-ups of up to 2000 times are achieved over low-level simulation.

Worst-Case Peak Temperature of Multi-Core Systems. The second main contribution of this thesis is an extension of the framework to calculate the worst-case peak temperature for single-processor systems [15] to many-core systems. After reformulating the task to an optimization problem, we propose an algorithm that solves the problem for periodic streams with bounded jitter and bursts. By performing experiments with a prototype implementation of our algorithm in MPA, we compare the framework with two approximation algorithms that offer significant speed-ups.

1.4 Outline

The remainder of this master thesis is organized in two parts. Thermal evaluation methods and appropriate techniques to obtain thermal characteristics by simulation are the topic of the first part of this thesis. Afterwards, the second part discusses the question of obtaining the worst-case peak temperature of a multi-processor streaming application using analytic worst-case methods.

The first part is organized in seven chapters. It starts with a short introduction to thermal simulation and a description of related work in Chapter 2. Afterwards, Chapter 3 introduces the concept of a thermal evaluation tool chain consisting of a high-level software synthesis tool, a low-level simulator, and a thermal analysis model. The viability of the proposed tool chain is assessed by a prototype implementation based on DOL, the MPARM virtual platform, and HotSpot. However, the tool chain seems inadequate for the thermal evaluation of several thousand candidate mappings as it is the case in design space exploration of a multi-processor streaming application.

Chapter 4 addresses this drawback by presenting the System Level Thermal Emulator (SLTE), an abstract thermal evaluation model to estimate the transient temperature evolution of a multi-processor streaming application in an early design stage. Compositional temperature evaluation models use timing and thermal parameters to model the behavior of a multi-processor streaming application. Chapter 5 presents a general approach for extracting these parameters by using a set of benchmark mappings to simulate the system on a low-level platform.

In Chapter 6, the viability of the proposed abstract model for emulating the transient temperature evolution, SLTE, is assessed by integrating a prototype implementation in DOL. Additionally, we discuss extensions for the thermal evaluation tool chain to carry out automated model calibration. In Chapter 7, we present three case studies, which are used to test and evaluate the prototype implementation of SLTE from Chapter 6. We discuss the speed-up and the accuracy of the proposed compositional temperature analysis model over the thermal evaluation tool chain. Finally, in Chapter 8, the proposed approaches for the thermal simulation are summarized and further work is listed.

The second part of this thesis is organized in four chapters and starts with an introduction to thermal analysis methods and an overview of related work. In Chapter 10, the task of computing an upper bound on the worst-case peak temperature of a multi-processor streaming application is reformulated as a multidimensional optimization problem. In this context, an algorithm to calculate this upper bound is proposed for applications, that only have periodic tasks with jitter. The approach to calculate the worst-case peak temperature is finally implemented in MPA. Afterwards, Chapter 11 presents various case studies to verify the viability of the proposed approaches. To improve the evaluation time, two approximation algorithms are discussed and evaluated. Finally, Chapter 12 summarizes the proposed methods and achieved evaluation results, and gives a review of further work.

In Chapter 13, the proposed approaches of this master thesis are summarized and compared in terms of their value for design space exploration of a multi-processor streaming application.

Appendix A collects additional case studies and supplementary material for the prototype implementation of SLTE. Appendix B presents the slides of the final presentation of this thesis and the mathematical symbols, which are used to formally describe the proposed models, are listed in Appendix C. To simplify the reading, Appendix D lists all acronyms introduced in this thesis.

Part I

Thermal Simulation

2 Thermal Simulation: An Introduction

2.1 Overview

Nowadays, reactive thermal management techniques are the preferred method to avoid overheating in microprocessors. However, these techniques can lead to a major loss in performance and complicate the prediction of the system behavior. One way to tackle this challenge is the use of system-level methodologies that guarantee both thermal constrains and real-time deadlines at the design time. Therefore, thermal evaluation methods are required to estimate the thermal characteristics of an application in an early design stage.

The first part of this master thesis addresses various techniques in conjunction with thermal simulation, that is, techniques that are based on simulating an application on a virtual or abstract platform. Although thermal simulation has various drawbacks over analytic best-case/worst-case methods [16], thermal simulation methods are still essential in the thermal evaluation for the following three aspects:

Transient Temperature Behavior. Analytic methods only calculate individual characteristics of an application, but they do not compute the transient evolution of the application. However, detailed investigations of the transient thermal behavior of an application are particularly required in a second design stage where typical questions include how long a specific temperature has been exceeded or how fast a system cools down.

Modeling Scope. While analytic methods are often restricted to a specific system model, simulation methods cover a much wider spectrum of specifications. For example, the analytic method to calculate the worst-case peak temperature of a single-node system that is presented in [15], only covers two power modes, however, real platforms often have a range that can be covered by the power dissipation.

Model Calibration. Compositional thermal evaluation models typically require a large number of thermal and timing parameters from different sources to accurately characterize a multi-processor system. Therefore, automated model calibration is required to extract the model parameters of such a system. As the low-level simulation is often the only accurate source for most of those parameters, thermal simulation is required to extract them.

To tackle these three aspects, we first propose a thermal evaluation tool chain, that is, a tool chain where the transient temperature evolution of a streaming application specified in a system-level language can be simulated. However, as the use of the thermal evaluation tool chain is too expensive in terms of execution time to be included in design space exploration, we propose the System Level Thermal Emulator (SLTE) in Chapter 4. It computes the temporal temperature evolution of each hardware component when the system specification and corresponding timing and thermal parameters are given. Based on the required timing and thermal parameters of SLTE, we discuss in Chapter 5 an approach to use the thermal evaluation tool chain to automatically calibrate such models. Afterwards, in Chapter 6, a prototype implementation of SLTE in the Distributed Operation Layer (DOL) is described, that is used to evaluate SLTE in Chapter 7. In three case studies, the advantages and drawbacks of the proposed approaches are discussed.

2.2 Related Work

Thermal management becomes a crucial task in modern integrated circuits as the areal heat dissipation has increased with submicron and three-dimensional integrated circuit design. Nowadays, hot spot heat fluxes can be up to 250 W/cm^2 [17] and jeopardize the reliability and timing guarantees of real-time applications. Puttaswamy et al. [18] investigated the thermal behavior of a high-performance microprocessor build in a three-dimensional technology and reported increases of the maximum temperature by 33 °C compared to a planar Integrated Circuit (IC). Therefore, in recent years efficient thermal management methods have been a hot topic in research. For example, in [19], a convex optimization technique for temperature-aware frequency assignment is proposed. Various architectural-level techniques for thermal management like Dynamic Voltage and Frequency Scaling (DVFS) and the stop-go policy are compared in [20]. Coskun et al. evaluated various thermal-aware job scheduling techniques in [21, 22]. In particular, the focus of [21] is on thermal management techniques with unknown workload like load balancing or a temperature-aware random policy scheduling. Dynamic thermal-aware job scheduling techniques for three-dimensional systems are presented in [22]. ThreshHot, a technique to reduce the number of thermal trespasses by selecting at each step the hottest job that does not exceed the thermal threshold, is proposed in [23].

Thermal management on the architectural or even on the Operating System (OS)-level has the drawback that real-time constraints cannot be guaranteed. Sabry et al. [24] proposed compilation techniques to reduce the percentage of hot spots based on a register allocation technique. A mixed-integer linear programming formulation to reduce the peak temperature of an application by guaranteeing hard real-time constraints is proposed in [25]. In [26, 27], temperature-aware task allocation and scheduling algorithms for embedded Multiprocessor System-on-Chip (MPSoC) are explored by comparing various thermal-aware with power-aware heuristics. All these approaches show that both the performance and the thermal analysis are crucial tasks in the design of streaming applications for modern two- and three-dimensional MPSoCs. However, there is currently a lack of tools for fast and accurate thermal analysis during early design stages of many-core streaming applications. Modern design flows for applications modeled as Kahn Process Network (KPN) or Synchronous Dataflow (SDF) enable the efficient design of multi-processor streaming applications for MPSoCs by providing automatic mapping of applications onto multi-processor platforms [3] and enable the reuse of the application for many different platforms. Artemis [28] or DOL [4] are two examples of such multi-processor software design flows and have in common that they support system-level performance analysis but have a lack in thermal analysis options. Model calibration is used to couple DOL with Modular Performance Analysis (MPA) [29] to provide a complete tool chain to analyze the performance of a multi-processor streaming application in an early design stage.

Simulating the thermal behavior of a multi-processor streaming application in a low-level simulator is the most obvious approach for thermal analysis during early stages of the design flow. In recent years, both software as well as hardware simulators for MPSoC platforms have been developed. The Multiprocessor ARM (MPARM) virtual platform [13] is an example of a cycle-accurate simulator in low-level SystemC [30] that emulates a multi-processor ARM architecture and includes a detailed power model [31]. Wattch [32] is another example of an architectural-level power-analysis framework to obtain the power dissipation of a modern platform. Furthermore, in [33], an architecture-level power analysis framework is presented that models the processor pipelines as networks to improve speed-up and accuracy. However, the slow evaluation speed of software simulators makes their use during design space exploration almost infeasible. Nowadays, hardware emulators are viable alternatives. In [34], Field Programmable Gate Array (FPGA) prototyping is used to speed-up co-verification of pure software simulators. In [35], a HW/SW emulation framework is proposed where an FPGA emulator is used to model the hardware components of a considered MPSoC to speed-up the evaluation time.

Estimating system temperature is typically based on thermal simulation or steady-state analysis. The later is based on approximating the thermal behavior by its steady-state temperature [36]. However, it can easily be shown that the use of the steady-state temperature might lead to wrong assumptions about the thermal behavior of an application. HotSpot [14] is the most popular member of the group of thermal simulators and has been extended by various features like oil-silicon cooling [37]. Focusing more on liquid cooling of three-dimensional ICs, 3D-ICE [17] is another architecture-level thermal simulator. Due to the need to solve the thermal differential equation, the thermal simulation time is often a major bottleneck during design space exploration. Model reduction [38, 39] is a promising approach to reduce the overhead introduced by solving the thermal differential equation. Another approach is presented in [40] where the solution of the thermal differential equation is precomputed for estimating the actual temperature of a Pentium 4 processor and reducing the temperature by adjusting the task schedule of an OS.

All of the presented models and tools have in common that none of them directly calculates the thermal behavior of a multi-processor streaming application defined by its software synthesis specifications, but all of them only provide a specific task towards the estimation of the thermal behavior. In [41], a virtual platform is proposed for evaluating and testing power and thermal management solutions based on the full system simulation platform Simics [42]. While providing various analysis tools, the virtual platform has the drawback of the evaluation speed and its restriction to the Intel Pentium 4 platform. In [35, 43] a hardware/software emulation framework is extended by connecting an FPGA emulation platform with a software thermal analysis library running on a host computer calculating the current system temperature of the on-chip components at runtime. The drawback of this emulation framework is the necessity of the additional hardware and the involved costs.

3 Thermal Evaluation Tool Chain

When modeling the thermal behavior of a multi-processor system at system-level, it turns out that the typical specifications used in performance analysis including the application, architecture, and mapping specification, are inadequate. Additional thermal characterizations like the temperature increase of an architectural component while executing a specific process are required to completely describe the thermal behavior.

In this chapter, a thermal evaluation tool chain is introduced, that is, a tool chain where the thermal behavior of a streaming application specified in a system-level language can be simulated. Finally, because the thermal evaluation tool chain is too expensive in terms of execution time to be included in design space exploration, an abstract, fast thermal evaluation model will be used instead. The proposed thermal evaluation tool chain will be mainly used for model calibration as well as to verify the abstract models proposed in Chapter 4. As a proof of concept, the proposed tool chain has been implemented using the Distributed Operation Layer (DOL) framework [4], the Multiprocessor ARM (MPARM) simulator [13], and the HotSpot thermal analysis model [14].

After presenting the concept of a thermal evaluation tool chain in Section 3.1, the MPARM simulator is briefly introduced in Section 3.2. The thermal simulator and the temperature-dependency of the power are discussed in Section 3.3 and the chapter concludes with a short summary.

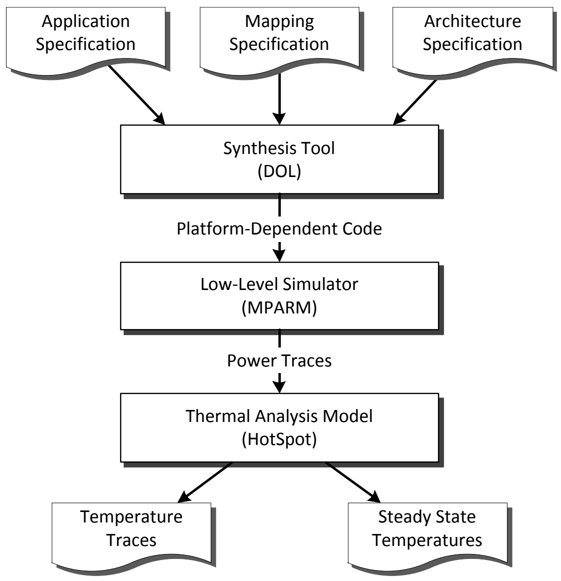

Application Specification
Mapping Specification
Architecture Specification
Synthesis Tool (DOL)
Platform-Dependent Code
Low-Level Simulator (MPARM)
Power Traces
Thermal Analysis Model (HotSpot)
Temperature Traces
Steady State Temperatures

Figure 3.1: High-level block diagram of the thermal evaluation tool chain.

3.1 Overview and System Specification

One way to extract the thermal characteristics of a multi-processor system is the use of a thermal simulation tool chain. In this chapter, the three step tool chain depicted in Fig. 3.1 is discussed. This section gives an overview of this thermal evaluation tool chain. First, its setup is described, while in Section 3.1.2, the system specification is presented in detail. Finally, Section 3.1.3 illustrates the thermal characteristics extracted from the tool chain.

3.1.1 Tool Chain

The transient thermal behavior of a process network is the result of a three step calculation. Based on the system specification, that is, the application, platform, and mapping specification, a synthesis tool generates the glue code of the application in the first step. In the second step, the application is executed in a low-level simulator to determine the transient power behavior of the application. The transient power behavior describes the temporal power consumption of the application while being executed on the specified architecture. In the last step, the transient thermal behavior, that is, the temporal temperature evolution, and the steady-state temperatures are calculated from the transient power behavior by a thermal simulator. The advantage of having three separate tools is that every tool can independently be upgraded as well as having the possibility to abstract each phase separately as we will show in Chapter 4.

Although the principles used for designing the thermal evaluation tool chain are not restricted to a specific synthesis tool, low-level simulator, and thermal analysis model, we use the DOL framework [4], the MPARM simulation platform [13], and HotSpot [14] to show the viability of our approach. The DOL framework introduced in Section 1.2.2 is based on a platform-independent programming model and supports the generation of glue code for multi-processor streaming applications on different platforms. Although many low-level simulation platforms are available such as Wattch [32], the ARM simulator proposed by Zhu et al. [44], or Sim-Panalyzer [45], to name only a few of them, the MPARM simulator [13] developed by the University of Bologna has the advantage that it provides an easy extendable statistic module [31] and is already coupled with DOL [10].

Timing information and power consumption of the process network are extracted from MPARM, the technical details of which are explained in Section 3.2. Both the dynamic and static power consumption of every microelectronic component are determined separately in the low-level simulation platform. The dynamic power has its source in charging and discharging various load capacitances of a CMOS device and is independent of the temperature. As the leakage current, the main source of the static power consumption, is temperature-dependent, the static power consumption is dependent on the current temperature of the system, as well. Therefore, the advantage of separating dynamic and static power is given by the ability to determine the actual, temperature-dependent static power in the thermal analysis model as explained in Section 3.3.3. The low-level simulation platform computes the static power consumption for a reference temperature, that is fixed in advanced and known by the thermal simulator. The granularity of the components can individually be selected based on the analyzed problem. Nonetheless, it turned out that a fragmentation in CPU cores, data and instruction memories, and buses provides a good balance between accuracy and computation speed.

The power characteristic of an application does not capture localized heating and the time dependency of the heat flow. Thus, thermal simulation is used to determine the actual thermal behavior. In our proof of concept implementation of the thermal tool chain, this task is performed by HotSpot [14], a thermal analysis model developed by the University of Virginia. As the thermal behavior does not only depend on the power characteristic of an application but also on the ambient temperature, the processor layout, and the heatsink configuration, HotSpot also requires additional input files, namely the floor-plan of the Integrated Circuit (IC) and the package configuration. The DOL framework is extended to automatically generate the floor-plan according to the architectural specification. Nonetheless, for more accurate simulation results, the floor-plan ought to be refined based on the real chip

Listing 3.1: Process network file used to specify the structure of a streaming application in DOL. The connection elements between the channels and processes are missing in this sketch.

```
<process name="generator">
    <port type="output" name="10"/>
    <source type="c" location="generator.c"/>
</process>

<process name="square">
    <port type="input" name="0"/>
    <port type="output" name="1"/>
    <source type="c" location="square.c"/>
</process>

<process name="consumer">
    <port type="input" name="100"/>
    <source type="c" location="consumer.c"/>
</process>

<sw_channel type="fifo" size="10" name="C1">
    <port type="input" name="0"/>
    <port type="output" name="1"/>
</sw_channel>

<sw_channel type="fifo" size="10" name="C2">
    <port type="input" name="0"/>
    <port type="output" name="1"/>
</sw_channel>
```

layout. Additional information about the thermal simulator is given in Section 3.3.

3.1.2 System Specification

The system specification of a Multiprocessor System-on-Chip (MPSoC) includes the application, architecture and mapping specification. It is the task of the synthesis tool to generate from the given system specification the glue code for executing the streaming application on the selected platform. The DOL framework includes a synthesis tool to generate code to be executed on MPARM out of the DOL specification [4].

Application Specification. The model of computation used in this project is the Kahn Process Network (KPN) [9], which has the advantage that the computation is separated from the communication and therefore, the application behavior is specified independently of the process network [3, 46]. An XML representation has been selected to syntactically describe the coordination of the process network. In Listing 3.1, such a process network file

Listing 3.2: Simplified code segment of the *square* process to describe its functionality. The corresponding process network is outlined in Listing 3.1.

```
void init() {
}

int fire() {
    float i;

    DOL_read((void*)PORT_IN, &i);
    i = i*i;
    DOL_write((void*)PORT_OUT, &i);

    return 0;
}
```

is outlined. A *sw-channel* describes the connection between two processes, which are defined by a *process* block. In a KPN, each process has its own processing characteristics while the overall functionality of the application is the combination of them. The *source* tag in the coordination file defines the source code file of a process, which is written in the host language, that is, in our case plain C/C++.

The coding rules to model the processing characteristics of a process are sketched in Listing 3.2. A process has to implement the `init()` and the `fire()` procedure. The former is executed once during the initialization of the application and its purpose is comparable with the one of a constructor. The `fire()` procedure is called repeatedly and contains the functionality of the process. Communication between processes is supported by dedicated primitives. In particular, `DOL_read()` is used to invoke a blocking read and `DOL_write()` is used to perform a blocking write.

Platform Specification. The information about the underlying architecture is summarized in the platform specification. Targeting the system-level design of an MPSoC, the platform specification is provided in an abstract model. The granularity depends on the requirements of the software synthesis framework. However, a typical system specification often includes the basic processing components, their attributes like the clock speed, and the way the components are connected. A custom-based XML format is used for the platform specification.

Mapping Specification. The specification of where and how a process is executed on an MPSoC is defined as mapping and consists of two parts. The binding defines the mapping of processes to processors, while the scheduling defines the mapping in the temporal domain, that is, the scheduling policy of each resource. Again, XML is used as format for the mapping specification.

3.1.3 Thermal Specification

The thermal evaluation tool chain calculates the thermal behavior of an application, and is used to record different thermal characteristics depending on the scope of application of the evaluation tool chain. To name only a few of them, the temporal temperature evolution, the steady-state temperature, and the maximum temperature of potential hot spots are often of interest. The temporal temperature evolution describes the temperature of a component at specific events in time and the steady-state temperature refers to the thermal behavior of a system when the transient power behavior recurs periodically. In our implementation of the tool chain, the temperature is recorded on a regular basis and the steady-state behavior of the system is calculated. Because we aim at estimating the temperature of several hundreds of design alternatives, in one design space exploration loop, we will use the proposed tool chain only to extract thermal parameters and calibrate faster analytic models with these realistic parameters.

3.2 From Execution to Power Traces

In our prototype implementation of the thermal evaluation tool chain, we use MPARM [13] as low-level simulator that provides the transient power behavior of a multi-processor streaming application. For this purpose, we extended the statistic and power modules of MPARM with the ability to compute power traces. The remainder of this section is organized as follows: First, the MPARM virtual platform is briefly introduced. Afterwards, in Section 3.2.2, the MPARM power models are presented in detail. The calculation of the transient power behavior is described in Section 3.2.3 and Section 3.2.4 presents the generation of glue code by the DOL synthesis tool to run streaming applications in the MPARM simulator.

3.2.1 MPARM Virtual Platform

MPARM [13] is a cycle-accurate virtual platform for emulating an ARM-based MPSoC. Originally developed by the University of Bologna to simulate multiple ARM processors connected by a simple bus, it has continuously been extended, and it is currently adapted to supports modern 3D ICs. An overview of the MPARM multi-processor platform is outlined in Fig. 3.2. The reconfigurable platform is composed of a variable number of identical 32-bit ARM processors, which are connected by a shared bus. Optionally, shared memories and Direct Memory Access (DMA) devices can be connected to the bus. SystemC [30] is used as modeling and simulation environment.

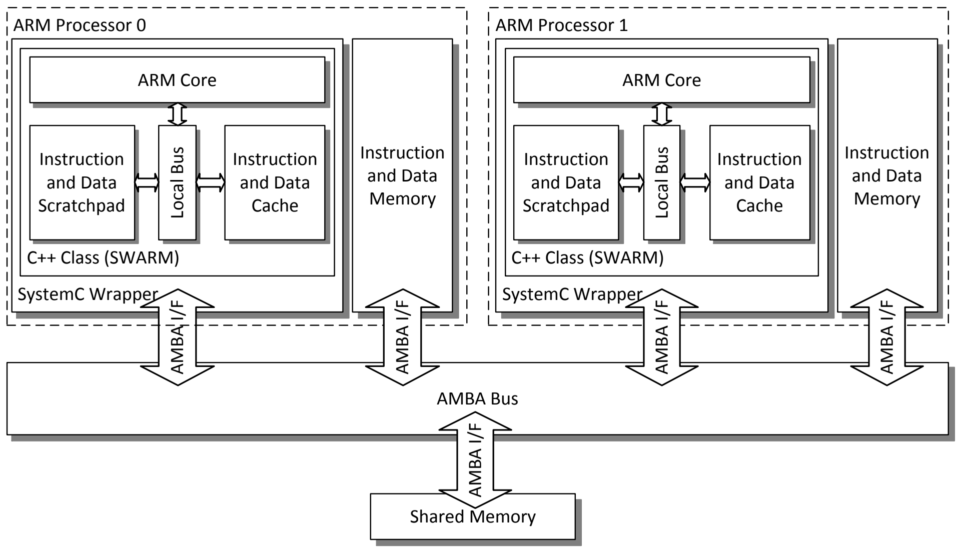


Figure 3.2: High-level schema of the MPARM architecture. The number of processors and shared memories can be individually adjusted.

Processing Elements. The most time-consuming part of a multi-processor simulation platform are the processing elements. Therefore, they have to be implemented in a fast and accurate way. Instruction-Set-Simulators (ISSs), which are often written in C/C++, provide a good balance between performance and accuracy. SystemC wrappers are used to connect the cycle-accurate communication architecture of the simulation platform with the coarse granularity domain of an ISS.

In MPARM, the default processing elements are ARM7 cores [13], which are modeled using the open source Software ARM (SWARM) simulator [47]. The SWARM simulator is basically written in C++ and communicates with its host using a `cycle` function that performs one clock cycle. Therefore, SystemC wrappers are used to embed the SWARM simulator into the SystemC environment. Note that the modular structure of the MPARM simulator allows an easy exchange of the SWARM simulator with another processor simulator.

Advanced Microcontroller Bus Architecture (AMBA) Bus Model. The shared bus used in the MPARM architecture is modeled as an AMBA bus. This is a widely used bus model for on-chip buses in MPSoC designs. AMBA supports scalability by the ability to plug-in multiple masters and slaves through proper bus interfaces and supports multi-master communication, which is indispensable for multi-processor systems.

Memories. MPARM supports three hierarchies of memories: scratchpad, caches, and main memory. As outlined in Fig. 3.2, every processing module has its own scratchpad, instruction cache, and data cache. They are directly connected to the core through the local bus of the processing element. Additionally, there exists an instruction and data memory for every processing element. However, these are only connected to the shared bus as slave devices, there exists no direct connection between local memory and its core. Furthermore, a various number of shared memories are supported, connected to the shared bus as slave devices, as well.

Operating-System Support. The real-time operating system Real-Time Executive for Multiprocessor Systems (RTEMS) [48] is running on top of the MPARM platform to provide high-level support for applications. Multiprocessor streaming applications can easily be implemented in RTEMS, because it provides both preemptive scheduler and message queues.

3.2.2 Power and Energy Statistics

MPARM implements a multifunctional statistics collector that has been extended to support energy and power statistics [31]. Each component is connected with a power model that calculates the energy consumption of every execution cycle. In its original configuration, the statistic collector supports just the record of the total energy consumption, and the calculation of the average power consumption of each component.

Overall Structure. The overall structure of the power model is outlined in Fig. 3.3. The *Statistic Collector* module provides multiple methods to handle and record events [13]. It analyzes every operation performed by a component and records timing, structural, and energy statistics of this operation. The later option is supported by attaching a *Power Model* [31], that provides the energy spent by the corresponding operation on a cycle-by-cycle basis, to every component.

All components except the SWARM processor and the AMBA bus invoke the power model when the module is activated. This power model computes the energy consumption for the operation performed by the component in a cycle and returns this value. At this turn, the activated module forwards the energy to the *Statistic Collector* module as shown in Fig. 3.4(a). The energy collection of the ARM core is implemented in a different way and is outlined in Fig. 3.4(b). As the ISS does not run whenever the ARM core is stalling, the energy collection is invoked by a separate routine that is activated at each cycle and keeps track of the current state of the core. In this way, the

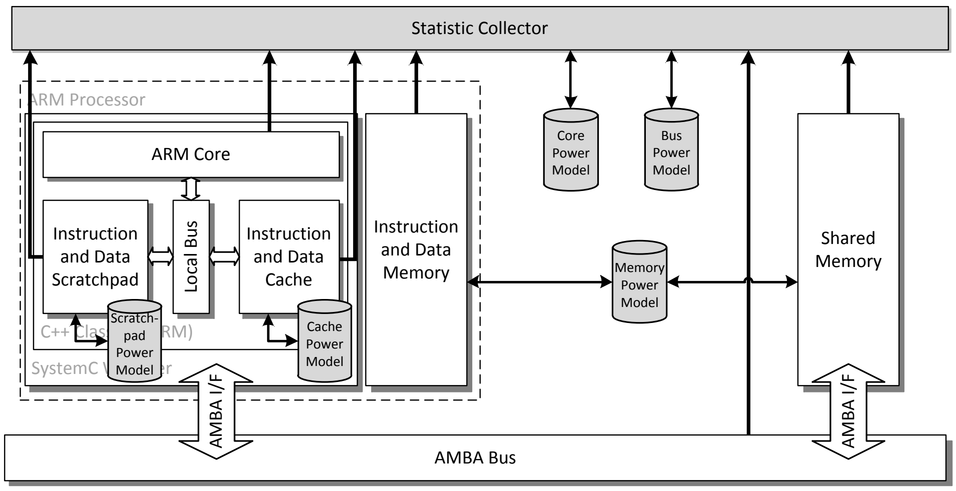
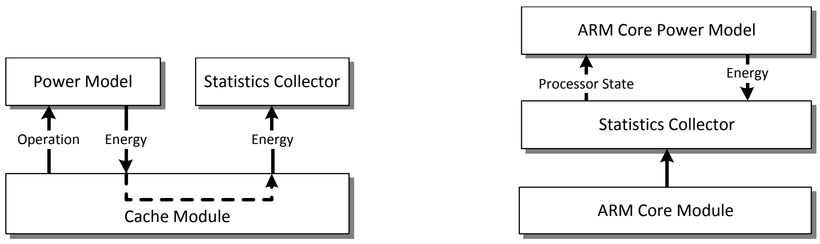


Figure 3.3: Overall structure of the MPARM power model. There exists a power model for each component, that calculates the energy consumption of an operation on a cycle-by-cycle basis.

energy spent in every cycle is provided. The energy collection for the AMBA bus is implemented similar to the one of the ARM core.

By summing up the energy of the individual invocations, the MPARM simulator collects the total energy consumed by every component. The simulator can also calculate the average power of each component by dividing the total energy by the total execution time. The MPARM simulator does not provide methods to output the energy consumption during the simulation, but only after the simulation has completed.



(a) Power model invocation for the memory modules.

(b) Power model invocation for the ARM core module.

Figure 3.4: The two implementations of power model invocation used in the MPARM virtual platform.

Power Models of the Individual Components. In the following, the different power models of the components used in this project are described. A 0.13 μm technology is assumed in all models.

- *SWARM Core.* The power model of the SWARM processor differs between the states *RUNNING*, *STALLED*, and *IDLE*. *IDLE* can only be reached using a software interrupt, and has the smallest power consumption of all states. The state *STALLED* models the situation when the instruction pipeline is blocked. The state *RUNNING* is used in all other cases. The individual energy values are obtained from an ARM7 processor on a 0.13μm technology [31]. While the *RUNNING* state consumes 0.055 mW/MHz, *STALLED* and *IDLE* states consume only 66% and 10%, respectively, of the energy consumed in the *RUNNING* state.

- *Memory.* The power model of both the caches and the private memories are based on an empirical model derived from the interpolation of the data extracted from a memory generator by STMicroelectronics [31]. The model differs between the four operations *READ*, *WRITE*, *STALL* and *NOP*. Furthermore, the energy consumption is a parametric value varying with the size of the memory.

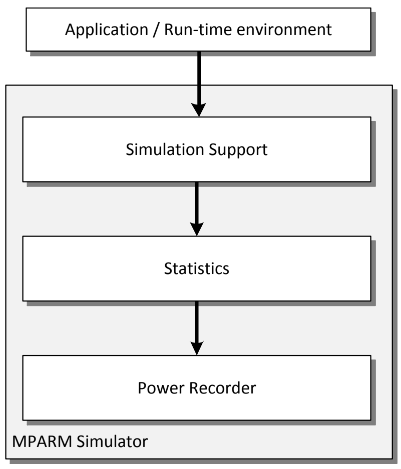
- *Cache.* The cache is modeled by two distinct cell arrays, for data and tag memory, with different energy consumptions. Additionally, the energy consumption depends on the cache type, and the line size of the cache. A more detailed description of the power model for the cache can be found in [31].

- *Bus.* The energy consumption of the AMBA bus is modeled according to [49] and is parameterized by the number of packets that are in transit on the bus.

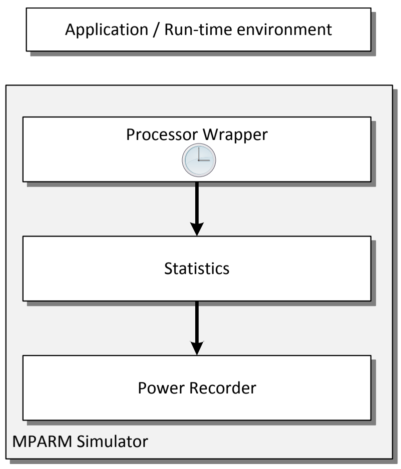
3.2.3 Power Traces

The calculation of the temporal temperature evolution of an application requires the knowledge of the transient power behavior of the system. This subsection summarizes the extension of the MPARM virtual platform to sample the power consumption on a regular basis.

Sampling the Power Consumption. The MPARM virtual platform already consists of a simple model to calculate the energy and power consumption of a system as described in the previous subsection. Nonetheless, it does not provide the functionality to record the transient power behavior.



(a) Event-based power trace sampling where the events are specified by the run-time environment.



(b) Power trace sampling on a regular basis. Therefore, no connection between the runtime environment and the simulator is necessary to sample the power.

Figure 3.5: Sampling trade-offs to generate power traces.

Using the total energy consumption W at time t_{i-1} and t_i, the average power consumption $P_{avg}(t_i, t_{i-1})$ of the period $\Delta t_i = t_i - t_{i-1}$ can be expressed as:

$$P_{avg}(t_i, t_{i-1}) = \frac{W_{t_i} - W_{t_{i-1}}}{t_i - t_{i-1}} \tag{3.1}$$

When using a sufficiently small period Δt_i, the power trace $P(t_i)$ at time t_i can be approximated as

$$P(t_i) \approx P_{avg}(t_i, t_{i-1}) \tag{3.2}$$

Skadron et al. reported in [50] that the use of a sampling interval of 10 k cycles provides a good trade-off between overhead and precision.

Power traces can either be sampled at specific events or on a regular basis. Sampling the power traces at specific events requires runtime environment support for invoking the simulator to record the power consumption whenever such an event occurs. The runtime environment has only the ability to perform such commands during the communication process or at the beginning of a process invocation. This sampling method is outlined in Fig. 3.5(a) and has the advantage that the original structure of the MPARM simulator is not modified and that the runtime environment controls the number of sampled power traces. One of the disadvantages is the limitation of the sampling frequency, in particular for computational intensive applications. Furthermore, the execution of additional commands in the runtime environment might reduce the performance of the application and thus, affects the statistics.

```
  time,  core_0, iCache_0, dCache_0, scratch_0, iScratch_0, ...
 20000, 8.6273,   7.1619,   1.8557,    0.0000,     0.0000, ...
 40000, 8.5663,   6.2899,   1.5568,    0.0000,     0.0000, ...
   ...,    ...,      ...,      ...,       ...,        ..., ...
460000, 8.8297,   8.2180,   1.8033,    0.0000,     0.0000, ...
480000, 8.7763,   7.9730,   1.7606,    0.0000,     0.0000, ...
500000, 8.8297,   8.2018,   1.8012,    0.0000,     0.0000, ...
520000, 8.8662,   8.5722,   1.9026,    0.0000,     0.0000, ...
540000, 8.9993,   9.6324,   1.3115,    0.0000,     0.0000, ...
560000, 8.8981,   8.4545,   1.3346,    0.0000,     0.0000, ...
580000, 8.8897,   7.9940,   1.1514,    0.1183,     0.0000, ...
600000, 8.6198,   6.5693,   1.0055,    0.0000,     0.0000, ...
620000, 8.9984,   9.8187,   1.3155,    0.0098,     0.0000, ...
640000, 8.9478,   9.3617,   1.4531,    0.0000,     0.0000, ...
660000, 8.8728,   8.4902,   1.3827,    0.0000,     0.0000, ...
680000, 8.9927,   9.0625,   1.4188,    0.1183,     0.0000, ...
700000, 8.6170,   6.6372,   0.9385,    0.0000,     0.0000, ...
720000, 9.1239,  10.7169,   1.4838,    0.0197,     0.0000, ...
```

Figure 3.6: Sketch of a power trace file as it is generated by the MPARM simulator. The first column contains the end time in nanoseconds of the interval that was used to record the power consumption. Every other column contains the power trace of one component.

To sample the power traces on a regular basis, a specific statistic method has to be invoked periodically as outlined in Fig. 3.5(b). As several units might operate at different frequencies, we need to track progress in terms of absolute time and not of simulation cycles. Recording the power traces on a regular basis has the advantage that it does not affect the simulated execution time of the application, thus does not influence the statistics. Furthermore, sampling the power traces at a sufficiently small interval, that is, every few thousands cycles, ensures that all relevant events are recorded. Disadvantages might include the overhead in generating data when the sample interval is selected to be much smaller than the time span between switches of the processor state. As the periodic sampling method does not influence the statistics and provides a higher resolution of the power consumption, we extended the MPARM simulator with this periodic power sampling module. In Fig. 3.6, an example of a power trace file is sketched. The first column includes the end time of the interval that was used to record the power consumption. Every other column contains the power trace of one component.

Static Power Consumption. In Section 3.3.3, the temperature-dependency on the power consumption will be discussed in detail. As will become apparent in the next section, the static power consumption highly depends on the temperature of the system, while the dynamic power consumption is temperature-independent. However, as the MPARM simulation platform has no knowledge of the temperature, the static power consumption can only be calculated for a reference temperature. Later, the thermal analysis model is able to calculate the static power consumption for the actual temperature of a component using the knowledge of the reference temperature.

This requires that the transient evolution of the dynamic and the static power is separately calculated by the MPARM virtual platform. Furthermore, as the power model of the ARM core is the only power model that provides support for the calculation of the static power consumption, we have extended all other power models to calculate their static energy consumption, as well. For this purpose, the overall power consumption is split into dynamic and static power consumption by using the ratio of the static power to the dynamic power for a predefined temperature as explained in Section 3.3.3.

3.2.4 DOL Software Synthesis for MPARM

Recently, DOL has been extended to support the MPARM simulation platform as target architecture [10]. Thus, applications coded as DOL specifications can automatically be executed on the MPARM virtual platform by the step that we call automatic software synthesis. Software synthesis is the generation of glue code to enable applications described at system-level in DOL to use services provided by the runtime environment executed on MPARM.

The fact that DOL is based on KPN model of computation has the advantage that a runtime environment for the MPARM simulation platform only needs to provide two basic functionalities: multi-processing for each separate core and the software channel implementation. Related to the multi-processing feature, as a process network is completely data-driven, a global clock and thus, global scheduling is not required. Instead, for processes sharing the same processor, local scheduling is provided by the multi-tasking feature of the RTEMS Operating System (OS) [10]. Related to the implementation of software channels, we distinguish between different software channel communication implementations for processes mapped onto the same tile, and for processes mapped onto different tiles. In addition to the classic message passing approach using shared memory, the First-In First-Out (FIFO) buffers can be allocated in the local scratchpad of the sender process as well as the one of the receiver process or even be split into two parts allocated in both the local scratchpads of the sender and receiver.

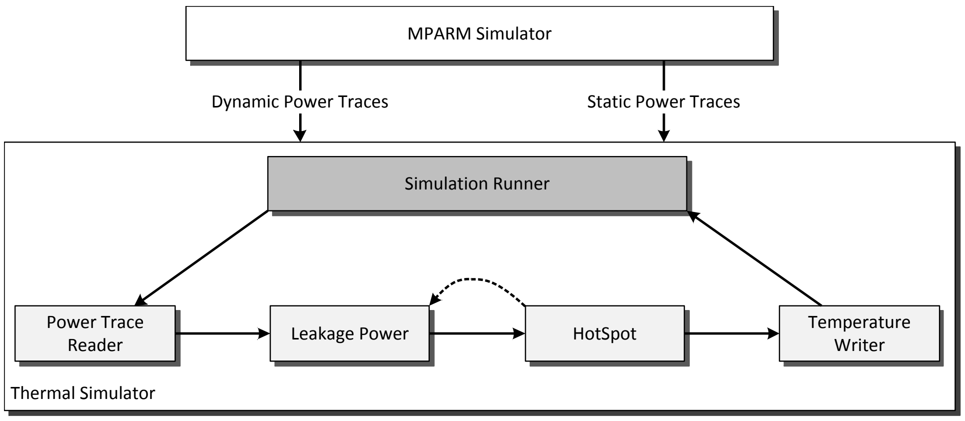


Figure 3.7: High-level block diagram of the thermal simulator used in the thermal evaluation tool chain to compute the transient thermal behavior of an application.

3.3 From Power Traces to Temperature Traces

The characterization of a system only on the basis of its power consumption has two disadvantages: It does not capture local heating and its non-linear behavior. Therefore, we need to calculate the temporal temperature evolution of an application using a low-level thermal simulator. HotSpot [14], a thermal analysis model developed by the University of Virginia, is used to calculate the temperature evolution in our thermal evaluation tool chain. This section details the thermal evaluation and is structured as follows: First, an overview of the thermal simulator is given by introducing the various modules of the thermal simulator. Afterwards, in Section 3.3.2, the used thermal analysis model, that is, HotSpot, is presented in detail. Finally, the temperature dependency on the static power is studied in Section 3.3.3.

3.3.1 Overview

The thermal simulator is after the synthesis tool and the low-level simulator, the third and last part of the thermal evaluation tool chain and is used to calculate the transient thermal behavior of an application. A high-level overview of the thermal simulator is given in Fig. 3.7. The input of the thermal simulator includes the power consumption, that is, the traces of both the dynamic and static power consumption, the floor-plan of the IC, and the package configuration. The MPARM simulator calculates the static power traces for a reference temperature that is well-known by the thermal simulator. This facilitates the thermal simulator to adjust the static power consumption to the current temperature of the specific microelectronic component. Besides HotSpot, the thermal analysis model to carry out most of the

```
  time,  core_0, iCache_0, dCache_0, scratch_0, iScratch_0, ...
 20000, 318.160,  318.154,  318.151,   318.151,    318.150, ...
 40000, 318.168,  318.158,  318.152,   318.152,    318.150, ...
   ...,     ...,      ...,      ...,       ...,        ..., ...
340000, 318.244,  318.208,  318.172,   318.172,    318.165, ...
360000, 318.247,  318.211,  318.173,   318.174,    318.166, ...
380000, 318.251,  318.215,  318.175,   318.175,    318.167, ...
400000, 318.254,  318.218,  318.177,   318.177,    318.168, ...
420000, 318.257,  318.221,  318.179,   318.178,    318.170, ...
440000, 318.260,  318.224,  318.180,   318.180,    318.171, ...
460000, 318.263,  318.227,  318.182,   318.181,    318.172, ...
480000, 318.266,  318.230,  318.184,   318.182,    318.173, ...
500000, 318.269,  318.234,  318.186,   318.184,    318.175, ...
520000, 318.272,  318.237,  318.187,   318.185,    318.176, ...
540000, 318.275,  318.241,  318.189,   318.187,    318.177, ...
560000, 318.277,  318.244,  318.191,   318.188,    318.178, ...
580000, 318.280,  318.247,  318.192,   318.189,    318.179, ...
600000, 318.282,  318.249,  318.194,   318.191,    318.181, ...
620000, 318.285,  318.252,  318.195,   318.192,    318.182, ...
```

Figure 3.8: Sketch of a temperature trace file as it is generated by the temperature writer of the thermal simulator. The first column contains the time stamp in nanoseconds, and every other column contains the corresponding temperature at that time. The time is provided in nanoseconds and the temperature in Kelvin.

computations, four additional modules are required to completely integrate the thermal simulator into the tool chain.

The *Simulation Runner* is responsible to control the thermal simulator. It calls the modules to calculate the alteration of the temperature by one power sample. The first module that the *Simulation Runner* calls, is the *Power Trace Reader*. After verifying the consistency of the dynamic and static power trace files, it transforms the power trace values in a format readable by HotSpot. The power traces are forwarded to the *Leakage Power* module to adjust the static power consumption to the temperature calculated in the previous iteration. The cumulative power consumption is calculated by summing up the dynamic and static power traces and is used as input of *HotSpot*. This thermal analysis model, described in detail in following subsection, is used to calculate the alteration of the temperature for one power sample. Finally, the *Temperature Writer* module writes the temporal temperature behavior as temperature traces to an output file for analyzing it with external tools and informs the *Simulation Runner* that the current iteration is completed. In Fig. 3.8, an example of such an output file is sketched.

Finally, after executing this loop for every power sample in order to generate the temporal temperature behavior, the *Simulation Runner* calculates the steady-state temperatures of the process network using the HotSpot library, as well.

3.3.2 HotSpot

In this project, HotSpot [14, 37], a fast and accurate thermal analysis model developed by the University of Virginia, is used to analyze the transient temperature behavior of our system. Validated using finite element simulation [50], many thermal-aware research projects (e.g., [20–22, 25, 26]) use HotSpot to calculate the thermal behavior of their system nowadays. This subsection briefly introduces the thermal model of HotSpot and illustrates the use of a thermal analysis model in a thermal evaluation tool chain.

Overview. HotSpot uses the well-known duality between the heat transfer and an electrical network [51] to calculate the temporal temperature evolution: the heat flow inside a microelectronic device is analyzed by considering a network of thermal resistances and capacitances. Using the floor-plan and the package configuration, it calculates an adjacency matrix that represents the *RC* circuit of the microelectronic components. The floor-plan describes the microarchitectural blocks and their layout. The package configuration defines the heatsink configuration, the ambient temperature, and various specific densities of the material used to manufacture the IC. In addition to classic heatsinks often used in desktop computers, HotSpot also supports natural convection in its newest version [37]. Natural convection is the process in which the fluid motion is only generated by the density difference in the fluid occurring due to temperature gradients.

HotSpot uses the transient power behavior as current sources in the equivalent electrical circuit and obtains the transient thermal behavior of the system by simulating the electrical circuit. By using the average power consumption, HotSpot also supports the calculation of the steady-state temperature. Although providing a simple interface to run as stand-alone application, HotSpot is mainly a model or library that can be used by other simulators to analyze the transient thermal behavior.

Model. The heat transfer can be investigated in analogy to an electrical phenomenon: The heat flow is modeled as current passing through a thermal resistance and the temperature difference is equivalent to the voltage [50]. In Fig. 3.9, an equivalent network to model the heat transfer of a single node is sketched. Thermal capacitances are used to model the time dependency of the temperature. Then, the thermal-aware circuit can be characterized

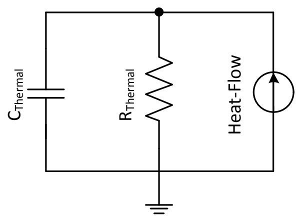
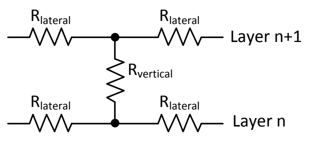


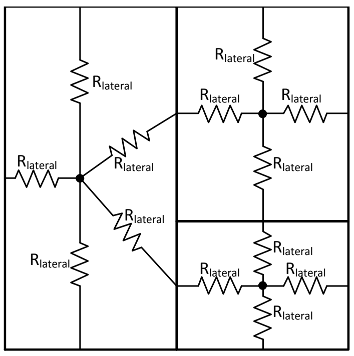
Figure 3.9: Equivalent *RC* network to model the heat transfer of a single node [14].

by its exponential rise and fall times in analogy to its thermal *RC* time constants. This duality between the heat transfer and an electrical circuit is the background of the thermal model used in HotSpot, which is presented in detail in the following.

The package used in most modern VLSI systems consists of several stacked layers made of different materials. Typical layers are the heatsink, the heat spreader, and the silicon die. Furthermore, each individual layer is subdivided into a number of blocks. Architecture-level units or regular grid cells can be used as substructures of the blocks. Each block is represented by one node in the *RC* circuit. The node is connected by a thermal resistance to the node of the next layer. To model the heat flow in lateral direction, each node is connected by a lateral resistance to the nodes representing the neighboring blocks in the same layer. The inter-layer model is outlined in Fig. 3.10(a), which sketches the side view of the stacked layer model. The two layers n and $n+1$ are connected by a thermal resistance $R_{vertical}$. A top



(a) Side view of the stacked layer model of HotSpot. The two layers n and $n+1$ are connected by a thermal resistance $R_{vertical}$.



(b) Top view of one layer of the thermal-model used in HotSpot. Each block is connected with its neighboring blocks by a lateral resistance.

Figure 3.10: Sketch of the stacked layer model used by HotSpot.

view of a single layer is given in Fig. 3.10(b). Three blocks are connected to each other by using lateral resistances.

A detailed description of calculating the values of the resistances and capacitances of a thermal *RC* model is given in [14], but the most important formulas are summarized next. The vertical thermal resistance can be computed by

$$R_{vertical} = \frac{t}{k \cdot A} \tag{3.3}$$

where t is the thickness, k the thermal conductivity of the layer, and A the cross-sectional area of the block. To calculate the lateral thermal resistance, one must account the heat spreading in lateral direction. Therefore, the resistance can be considered as the spreading/constriction thermal resistance of the neighboring part within a layer to that specific block [14]. A thermal capacitance is used to connect each node to the ground. The value of the thermal capacitance is given by

$$C_{th} = \alpha \cdot c_p \cdot \rho \cdot t \cdot A \tag{3.4}$$

where $\alpha \approx 0.5$ is a scaling factor [14], c_p and ρ are the specific heat and density of the material, respectively. Furthermore, the heatsink-to-air convection thermal resistance is modeled in HotSpot as

$$R_{convection} = \frac{1}{h \cdot A} \tag{3.5}$$

with h the heat transfer coefficient and A the surface area. Connecting all these resistances and capacitances results in a *RC* network that describes the transient thermal behavior of a modern IC.

3.3.3 Leakage Power

Nowadays, the power consumption is one of the biggest challenges in the design and production of CMOS devices and two power sources are distinguished: (1) dynamic power, that comes from charging and discharging various load capacitances, and (2) static power, also called leakage power, whose main source is the leakage current. As recently as a few years ago, it has been assumed that the only significant source of power consumption is the dynamic power. But by shrinking the processor's technology below 100 nanometers, the supply voltage is reduced and less dynamic power is generated. This causes that the static power begins to dominate the power consumption of modern CMOS devices [52].

Unlike the dynamic power, the static power of a CMOS device highly depends on its temperature. Fallah et al. [53] have shown that the static power consumption of a 15 mm die fabricated in a standard 0.1 μm technology is only 6% of its dynamic power consumption at 30 °C, but 33% at 80 °C and 56% at 110 °C. Thus, to accurately estimate the power consumption of a modern multi-processor system, the temperature-dependency of the static power has to be taken into account. The mechanism used in the thermal simulator to adjust the static power consumption to the system temperature is presented in detail in the following.

Power Consumption of CMOS Devices. The dynamic power depends on the total capacitance load C of all gates, the transistors supply voltage V, and the operational frequency f and can be expressed by

$$P_{Dyn} = C \cdot V^2 \cdot f. \tag{3.6}$$

The main source of the leakage power is the leakage current I_{Leak}, which leaks through the transistors even if they are turned off. Thus, the total power consumption can be expressed by the sum of the dynamic and leakage power:

$$P_{Tot} = P_{Dyn} + P_{Leak} = C \cdot V^2 \cdot f + V \cdot I_{Leak} \tag{3.7}$$

where P_{dyn} is expressed by (3.6). The total leakage current in a CMOS transistor results from four different sources [53]: the reverse-biased junction leakage, the gate induced drain leakage, the gate direct-tunneling leakage and, the subthreshold leakage. The later is the dominant component of the leakage power [53] as it is much larger than the other leakage current components. Thus, the total leakage current can be expressed as

$$I_{Leak} \approx I_{Sub}. \tag{3.8}$$

The subthreshold leakage is also called weak inversion leakage and can be computed by the following equation, again borrowed from [53]:

$$I_{Sub} = \frac{W}{L} \cdot \mu \cdot \nu_{th}^2 \cdot C_{sth} \cdot e^{\frac{V_{GS}-V_T-V_{off}}{\eta \cdot \nu_{th}}} \cdot \left(1 - e^{\frac{V_{DS}}{\nu_{th}}}\right) \tag{3.9}$$

where W is the transistor's width and L its length, μ denotes the carrier mobility, and $\nu_{th} = \frac{k \cdot T}{q}$ the thermal voltage at temperature T, where k is the Boltzmann constant and q is the charge of an electron. Furthermore, the summation of the depletion region capacitance and the interface trap capacitance is denoted as C_{sth}, η is the subthreshold swing coefficient, V_{GS} is the gate voltage, V_T denotes the threshold voltage, and V_{off} is an empirical BSIM3 parameter[1].

[1]BSIM3 is a MOSFET SPICE model developed by the University of California, Berkeley that is used for circuit simulation and CMOS technology development [54].

Temperature Dependent Leakage Power As recently as a few years ago, the leakage power was often neglected or assumed to have a small, constant value. However, recently, various methods have been developed to approximate the leakage power as it becomes the dominant factor in prospective CMOS devices. Heo et al. [55] first introduced a simple exponential model for the leakage power:

$$P_{Leak} = P_{Leak,0} \cdot e^{\beta(T_j - T_0)} \tag{3.10}$$

where $P_{leak,0}$ denotes the leakage power at temperature T_0, and β is a technology parameter.

By using the Taylor series expansion of the subthreshold leakage current, Liu et al. [56] expressed the leakage power as a linear function of the temperature. They could show that the linear model is fairly accurate in the normal operating temperature range of modern ICs. As a linear model has various advantages in theoretical research, the model is used in various other publications like [15, 25]. A different approach is used in [36]. By demonstrating that the exponential term $e^{V_{DS}/\nu_{th}}$ in (3.9) is negligible, they formulated the leakage power as a function with a quadratic dependency of temperature.

In this project, we are mainly interested in guaranteeing that the critical temperature of an IC is not exceeded as a discarding of this limit might result in a reduction of the clock frequency and real-time constraints cannot be met any more. Thus, our leakage power model should be applicable to all devices of our architecture and its calculation speed should be pretty fast in order to not excessively increase the execution time of our simulation. On the one hand, both the exponential method from [55] as well as the linear approach from [56] might not provide accurate temperatures at the boundaries of our temperature range. However, the upper bound of the temperature range is the most interesting region in the design space exploration. On the other hand, leakage power simulators like [57] provide accurate results for almost all temperature ranges but are often only applicable to specific devices like the cache.

In this project, we will use the method presented in [50, 58] to calculate the ratio between the dynamic and the leakage power as it is a good trade-off between speed and accuracy. A detailed explanation and derivation of the method can be found in [58] while the following paragraph gives a short overview of the method.

Leakage Power Model. Without introducing a significant error, we first assume that the supply voltage is much greater than the threshold voltage. After grouping the constants to a technology parameter, the subthreshold current from (3.9) can be rewritten as

$$I_{Sub} = K_1 \cdot T^2 \cdot e^{\frac{q}{k\eta}\left(\frac{V_{GS} - V_T - V_{\text{off}}}{T}\right)} \tag{3.11}$$

where K_1 is an empirical constant depending on the technology. Following the explanations in [58], we can express the subthreshold swing coefficient η as

$$\eta = S \cdot \frac{q}{K \cdot T \cdot \ln(10)} \tag{3.12}$$

where S is called the subthreshold slope. Then, the threshold voltage can be calculated by using the leakage current in its edge case:

$$V_T = \eta \cdot \frac{k \cdot T}{q} \cdot \ln\left(\frac{J_0}{I_0}\right) \tag{3.13}$$

where J_0 is called the saturation drive current and denotes the leakage current under the condition $V_{GS} = V_T$ and I_0 is the subthreshold leakage current, that is, the leakage current when $V_{GS} = 0$.

By assuming that the transistor is off and combining the technology-dependent voltages to a constant, the leakage current can be expressed as

$$I_{Leak} = K_1 \cdot T^2 \cdot e^{-\frac{K_2}{T}} \tag{3.14}$$

where

$$K_2 = \frac{q}{k \cdot \eta} \cdot (V_T + V_{\text{off}})\,. \tag{3.15}$$

Using the supply voltage V and (3.14), the total leakage power can be computed by

$$P_{Leak} = I_{Leak} \cdot V = K_1 \cdot V \cdot T^2 \cdot e^{-\frac{K_2}{T}}\,. \tag{3.16}$$

The dynamic power remains constant with respect to the temperature, thus the ratio R_T of the leakage power to the dynamic power at temperature T can be expressed by

$$R_T = \frac{R_0 \cdot P_{Leak,T}}{P_{Leak,T_0}} \tag{3.17}$$

where T_0 is the reference temperature, P_{Leak,T_0} the leakage power at temperature T_0 and R_0 denotes the corresponding ratio. Using (3.16), the above formula can be expressed as

$$R_T = \frac{R_0}{V_0 \cdot T_0^2} \cdot e^{\frac{K_2}{T_0}} \cdot V \cdot T^2 \cdot e^{\frac{-K_2}{T}} \tag{3.18}$$

where V_0 is the nominal voltage.

This ratio between the leakage power and the dynamic power is used in the thermal interface to calculate the temperature-dependent leakage power.

We assume to have a constant subthreshold slope of 85 mV/dec [58] and calculate the ratio between the saturation drive current and subthreshold leakage current using the values from the ITRS report for process integration, devices and structures [59]. The empirical BSIM3 parameter V_{off} is defined in [60] and $R_0 = 10\,\%$ is used as ratio at the reference temperature 85 °C [50]. Note that a 130 nm technology generation is assumed for all constants as the energy consumption used in the MPARM simulator are calculated with respect to this technology [31].

3.4 Summary

As the areal density of the heat dissipation increases rapidly in modern 2D/3D ICs, the thermal analysis of candidate mappings becomes indispensable in the design space exploration of a modern multi-processor streaming application. Hot spots might cause a processor to reduce its clock frequency and real-time constraints cannot be guaranteed any more. As the power characteristic of an application does not capture localized heating as well as its non-linear behavior, the temporal temperature evolution has to be analyzed.

In this chapter, a model of a thermal evaluation tool chain has been proposed. Three different tools are required to accurately model the thermal behavior of a multi-processor streaming application specified in a system-level language.

In the first step, a synthesis tool generates glue code that connects the application described at system-level with the services provided by the runtime environment of the low-level platform. In the second step, the transient power behavior of the system is determined by simulating the application in a low-level simulation platform. Finally, the temporal temperature evolution of the streaming application is calculated in the thermal simulator. After adjusting the static power consumption to the current temperature of the system, a thermal analysis model is used to perform the actual calculation of the temperature alteration. To show the viability of our proposed thermal evaluation tool chain, an implementation with the DOL framework, the MPARM simulation platform, and the HotSpot thermal analysis model is discussed.

4 System-Level Thermal Emulation

The process of determining the optimal mapping of a complex multi-processor streaming application often requires the evaluation of thousands of design alternatives. Therefore, the method for the timing and thermal evaluation of the design alternatives has a high influence on the execution time of the design space exploration. The use of the thermal evaluation tool chain presented in the last chapter seems inadequate as even the evaluation of a single design suggestion might take more than 24 hours as shown in [10].

In this chapter, we answer the question how to represent the thermal behavior of each hardware component when the system specification, that is, the application, architecture, and mapping specification, and the required timing and thermal parameters are given. For this purpose, we introduce the System Level Thermal Emulator (SLTE), a compositional temperature evaluation model for multi-processor streaming applications. The model consists of two parts. First, the application scheduling and the corresponding power consumption of the system are determined by a segment-based power annotation model. Afterwards, the temporal temperature evolution of the system is computed by an extended thermal simulator that performs compute-intensive calculations during the model calibration.

First, in Section 4.1, the considered problem is stated, along with an overview of the various abstraction levels for the thermal analysis. Afterwards, in Section 4.2, the proposed approach for the power model is presented in detail. A method to partly solve the thermal differential equation system in advanced is presented in Section 4.3 and finally, a summary concludes the chapter.

4.1 Overview

An efficient implementation of the design space exploration to find a time and thermal optimal mapping between a multi-processor streaming application and a distributed architecture requires the use of fast and accurate evaluation methods for timing and thermal analysis. Even though multiple approaches for measuring the timing behavior of a system have been proposed, there exists a lack of methods to analyze the thermal behavior of an application in an efficient manner. After giving a formal statement of the considered problem, the proposed compositional temperature evaluation model is summarized in this section. Its remainder is structured as follows: Section 4.1.1 defines the discussed problem while Section 4.1.2 introduces the methodology of compositional performance evaluation. Finally, in Section 4.1.3, the various levels of abstraction in the design of an abstract thermal model are summarized.

4.1.1 Problem Statement

Given the specification of a parallel application and a multi-core large scale architecture, we address the problem of finding the mapping, that is, the binding and scheduling of the application onto the architecture in a time and thermal optimal manner. Both the binding of application elements to computation resources as well as the scheduling policies and their parameters for each computation and communication resources are varied until the optimal mapping is found.

In order to reduce the number of binding and scheduling combinations that need to be analyzed in the design space exploration, evolutionary algorithm based approaches have been proposed as feasible and effective solutions [12]. Fast and accurate evaluation methods for system performance and temperature are key ingredients of this mapping optimization. Recently, many fast and accurate evaluation methods for the system performance are proposed, such as [29, 61, 62]. Consequently, in this project, we aim to develop a fast and accurate evaluation method for the temperature.

The use of a low-level simulator to measure the transient temperature evolution of an application is one possible evaluation method. Chapter 3 discusses this concept by using a low-level simulation platform in combination with a thermal analysis model. While providing accurate results for the transient power and temperature behavior of an application, the simulation method has the disadvantage of low evaluation speed. Huang et al. [10] have shown that simulating the MPEG-2 decoder [63] on the MPARM platform with five parallel tiles may take more than 5000 times its execution time. However, this does not even include the thermal analysis model. Thus, the use of

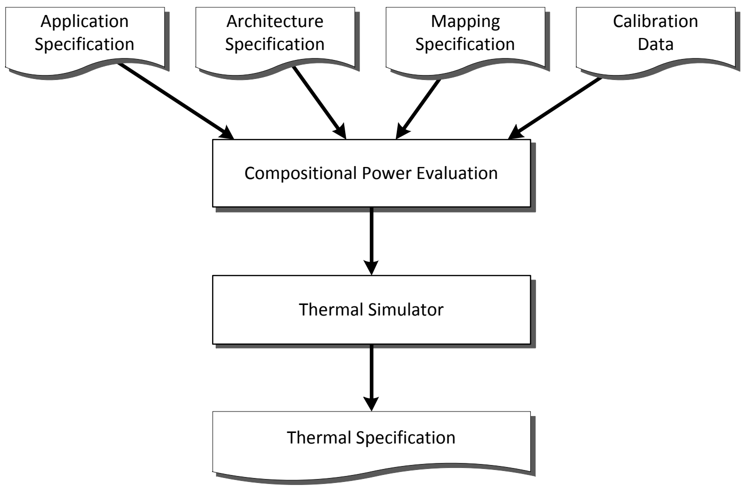

Application Specification
Architecture Specification
Mapping Specification
Calibration Data
Compositional Power Evaluation
Thermal Simulator
Thermal Specification

Figure 4.1: High-level block diagram of a compositional temperature evaluation model for extracting thermal characteristics of a many-core system.

low-level simulation methods for the computation of the optimal mapping in the design space exploration is infeasible and faster estimation methods are required.

Starting in this chapter, we discuss the problem of generating and calibrating a compositional temperature analysis model for accurate and fast design space exploration. We aim to achieve an evaluation time that is faster then the execution time of the corresponding application on the simulated platform.

4.1.2 Compositional Temperature Evaluation

Compositional evaluation models have been shown to be a good choice for performance analysis by providing accurate and fast estimations of the performance characteristics of a multi-processor system [29]. In compositional evaluation, subsystems are individually characterized and analyzed by combining pre-characterized evaluation components. When using compositional evaluation models for temperature characterization, a piece of software executing on a specific processor might be modeled for instance by its Average Case Execution Time (ACET) and the maximum power consumption of the processors subcomponents. A compositional evaluation model has the advantage that the characterization of each individual component needs to be performed only once. Afterwards, components can be processed as black boxes and combined to build the system.

Fig. 4.1 provides a high-level overview of the compositional temperature evaluation model proposed in this project. In addition to the system specification, that is, the application, architecture, and mapping specification,

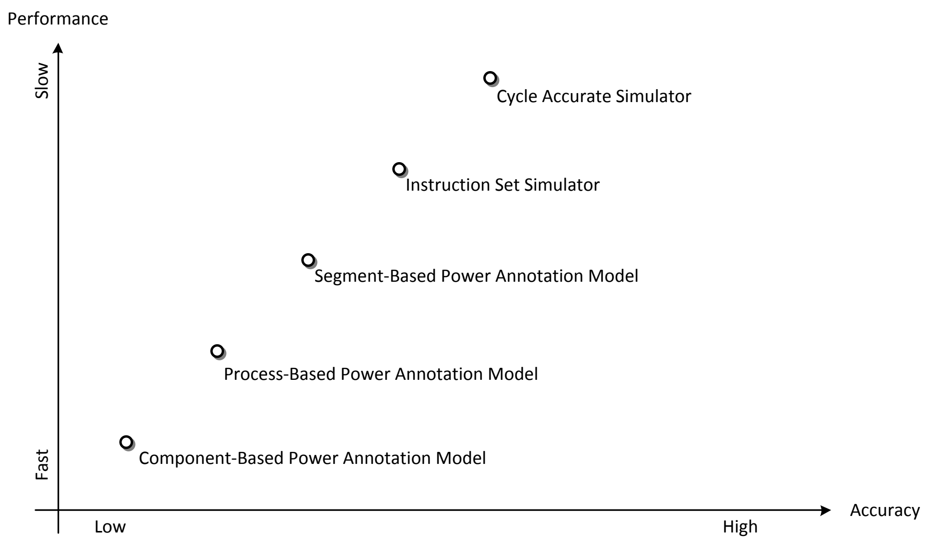


Figure 4.2: Overview of various levels of abstraction of designing a power evaluation model.

calibration data is used as input to the temperature evaluation model. The calibration data include thermal and timing parameters of the application. The process of performing the model calibration is presented in detail in Chapter 5, while various approaches for generating a thermal analysis model are discussed in this chapter.

4.1.3 Levels of Abstraction

Thermal evaluation models generated from synthesizable specifications can be designed at different levels of abstractions, in a trade-off between the model accuracy and performance. To increase the performance of an evaluation model, the specific behavior and characteristics of a system are required to be abstracted out. In the area of compositional evaluation, the level of abstraction is also in a strict correlation with the acquisition of calibration data, as different characteristics might be required depending on the granularity of the system. In this model, we mainly concentrate on abstracting the power consumption of an application.

In Fig. 4.2, various power consumption models are outlined and compared with respect to performance versus accuracy. While the use of a cycle accurate simulator and ISS promises relatively accurate power estimates, they suffer from low execution time. Therefore, they are mostly used at low-level for validating the final design and to extract performance data. On the other hand, compositional evaluation models are used at system-level where various designs have to be explored and compared, and fast decisions have to be taken. Based on the ideas of a compositional evaluation model, the segment-based power annotation model proposed in Section 4.2.3 to compute the power consumption of an application, offers a good trade-off between performance and speed. The simulation speed can further be increased either using a process-based or even a component-based power annotation model, as presented in Section 4.2.2.

4.2 Abstract Power Evaluation Models

In Section 4.1, we have shown that compositional evaluation models seem to be a good choice to model the transient power consumption of a multi-processor system at system-level. By abstracting out the power characteristics of a system as being constant for a specific time interval, the transient power consumption of the system can be calculated much faster than using a low-level simulator. While the concepts of the proposed models can be applied to any distributed system, we focus us in this chapter to applications modeled as KPN that are executed on distributed memory architectures.

In the following, three models of various granularities to approximate the power consumption of a multi-processor streaming application are introduced. This ranges from assigning a constant power value to each component, to segmenting a process in various small sequences of instructions, and each segment being annotated with the corresponding power and timing characteristics. Note that we are only interested in estimating the dynamic power consumption of a system. In this project we assume that the static power consumption of a component is independent of the instructions that are executed on the corresponding unit. This section is organized as follows: First, in Section 4.2.1, the problem is formally described. Afterwards, in Section 4.2.2, two simple models with block-based power annotation are presented, while finally, in Section 4.2.3, a power annotation model is presented where all processes are segmented in various small sets of instructions.

4.2.1 System Specification and Problem Formalization

As a general evaluation model does not exist, we restrict our model to consider KPN based applications executing on a distributed memory architecture. Although this class of systems is similar to the ones described in Chapter 3, we first formally describe this class of systems as well as the considered problem to establish a common language.

Formal System Specification. While Section 3.1.2 gives an overview of our high-level specifications of the system, this section introduces a formal definition of the system[1].

[1] An overview of all symbols used in this section is given in Appendix C.

- *Application:* We represent a KPN application as a directed, connected graph $\mathcal{A} = (V, Q)$. Every process $v \in V$ represents a node in the graph. Edges are used to model the unbounded FIFO channels $q \in Q$ that are used by processes to communicate.

- *Architecture:* The architecture $\mathcal{T}$ is assumed to be a homogenous multiprocessor system with distributed memory architecture. We call a heterogeneous processor subsystem a tile $\theta \in \Theta$. The architecture $\mathcal{T}$ consists of totally n homogenous tiles $\theta_i, i \in \{1, \ldots, n\}$. Each tile θ_i has m components $c_{j,i}, j \in \{1, \ldots, m\}, i \in \{1, \ldots, n\}$, $c_{j,i}$ is the jth component of the ith tile, and o shared components $sc_k, k \in \{1, \ldots, o\}$.

- *Mapping:* The mapping of the application onto the architecture consists of the binding b and the scheduling information s. The binding can be represented by a vector $b = (b_1; \ldots; b_{|V|}) \in \{1, \ldots, |T|\}^{|V|}$ where $b_i = k$ if process v_i is assigned to tile t_k.

- *Power Model:* Given an application specification $\mathcal{A}$ and an architecture specification $\mathcal{T}$, the power consumption of the component c when performing the process v is P_c. The power consumption of a tile θ with totally n components can be represented as a vector $P(t) = (P_1(t); \ldots; P_n(t))$.

- *Calibration Data:* The calibration data is represented by a set $\psi = (\psi_{\text{Time}}, \psi_{\text{Thermal}})$, where ψ_{Time} are timing parameters and ψ_{Thermal} thermal parameters.

Formal Problem Statement. Using the notations introduced above, the problem of estimating the power consumption can be stated as follows:

Given a system specification $S = \{\mathcal{A}, \mathcal{T}, (b; s)\}$ consisting of an application specification $\mathcal{A}$, an architecture specification $\mathcal{T}$, and a mapping specification $(b; s)$, generate the transient power characterization $P(t)$ of the system by using some pre-characterized calibration data ψ.

4.2.2 Block-Based Power Annotation Models

Generating an accurate power annotation model is a nontrivial task as both the thermal and temporal characterization of the system have to be considered. To illustrate the use of a compositional evaluation model, two simple power annotation models are presented in this subsection, namely the component-based and process-based power annotation model. While the former does not consider timing parameters, the later includes timing parameters to refine a compositional evaluation model.

Component-Based Power Annotation Model. First, a really simple model to perform thermal evaluation during design space exploration is introduced. The power consumption is computed in two steps. In the first step, a constant power value is calculated for every type of component. In the second step, the system specifications and the power values are combined to calculate the transient power behavior of the system by setting the power consumption of each individual component to the corresponding power value. For example, having a system with n cores, their transient power behavior can be expressed as

$$P_{\text{core}_1}(t) = P_{\text{core}_2}(t) = \ldots = P_{\text{core}_n}(t) = P_{\text{core}} = \mathit{const} \quad \forall t. \tag{4.1}$$

As the power values of the components only depend on the thermal parameters of the system but not on the mapping, the transient power behavior is the same for all mappings and therefore, the model is trivial and cannot be used to make mapping decisions in the design space exploration. However, this model can be used to quickly verify whether thermal analysis is necessary in the design space exploration or all design alternatives automatically fulfill the thermal requirements of the system, that is, the critical temperature of a component is never exceeded. With the peak power consumption of the components as power metrics, the models calculates an upper bound of the transient power behavior of the system and the following equation is always fulfilled:

$$P_{\text{system, c}}(t) \leq P_{\text{model, c}}(t) \quad \forall t, c \tag{4.2}$$

where $P_{\text{system, c}}(t)$ is the actual transient power consumption of a component c and $P_{\text{model, c}}(t)$ is the estimated upper bound on the transient power consumption of the component c using the component-based power annotation model. Obviously, this outcome can be used to calculate an upper bound of the transient temperature trace of the system, that is, the predicted temperature will always be equal or higher as the actual temperature of the system. Whenever all predicted temperatures are lower than the critical temperature, no thermal evaluation is required for this multi-processor streaming application.

As an example, the transient power behavior of a system with two homogenous tiles is outlined in Fig. 4.3. The core consumes 10 mW and the cache 7.5 mW. In the analyzed system, *process B* and *E* are mapped to *tile 0* while *process A*, *C*, and *D* are mapped to *tile 1*. In this naive model, obviously, the power consumption of every core is always 10 mW and the one of the cache 7.5 mW.

Process-Based Power Annotation Model. It is well known that the power consumption of a component depends on the actual instructions that

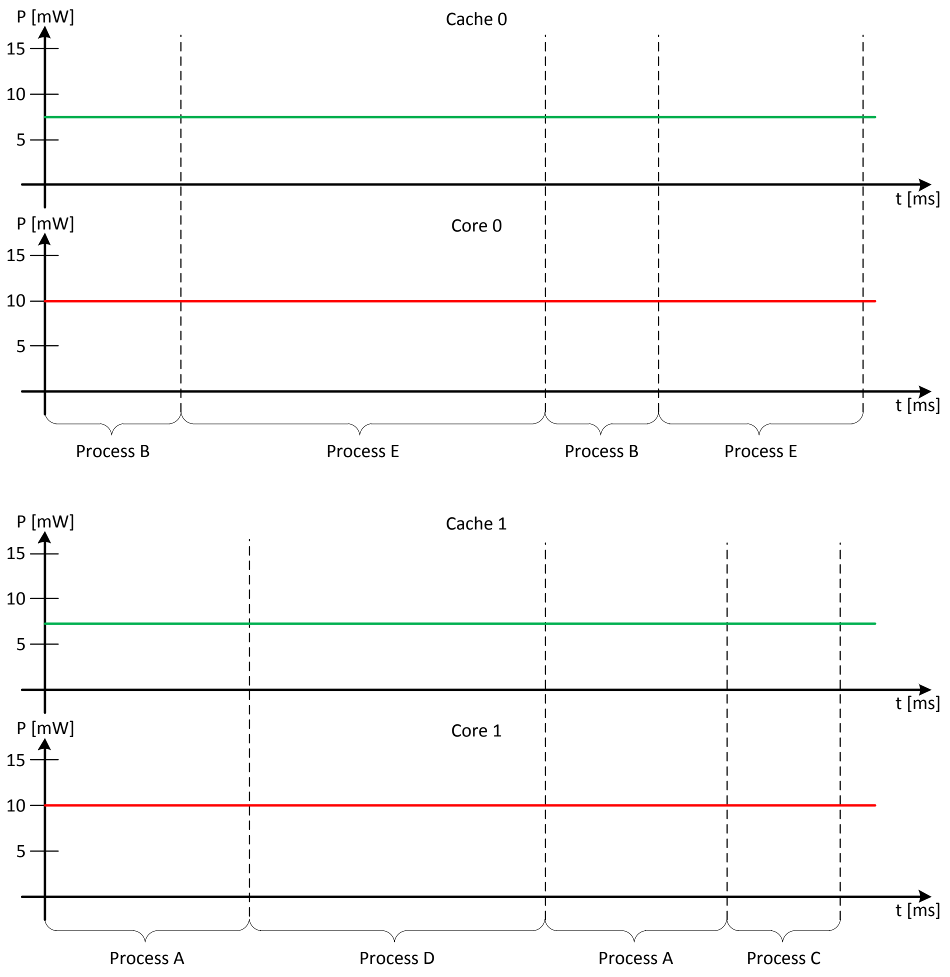


Figure 4.3: Example of the component-based power annotation model using a system with two homogenous tiles. The transient power behavior of every core and cache is constant all the times.

are executed. As all processes of a KPN often execute different instructions, the power characteristics of the processes differ from each other. To tackle this issue, the process-based power annotation model differs between individual processes to get a more accurate estimation of the transient power behavior of a system than the naive component-based model.

Similar to the component-based power annotation model, the generation of the transient power evolution is split into two steps. In the first step, there is a constant power value calculated for each process and type of component using the thermal parameters of the calibration data. In the second step, a process-based scheduling of the system is annotated with the power values to calculate the transient power characteristic. As an example, given that processes A and C are mapped to core i, the transient power behavior of core i can be expressed as

$$P_{\text{core}_i}(t) = \begin{cases} P_{\text{A, core}}(t) = const_1 & t \in t_A \\ P_{\text{C, core}}(t) = const_2 & t \in t_C \\ 0 & \text{else} \end{cases} \tag{4.3}$$

where $P_{\text{A, core}}(t)$ and $P_{\text{C, core}}(t)$ denote the power consumption of a core when processes A and C are executed for time spans t_A and t_C, respectively.

Compared to the component-based power annotation model, this model requires timing parameters to calculate the execution order of the individual processes. Trace-based simulation, as it is presented in [64], is one option to calculate the execution order of processes.

An example of the power traces generated by the process-based power annotation model is given in Fig. 4.4. The architecture consists of two homogenous tiles. *Process B* and *E* are mapped to *tile 0*, and *process A*, *C* and *D* are mapped to *tile 1*. The individual power values of different processes are outlined in Table 4.1.

Compared to the component-based power annotation model, the output of the process-based power annotation model is a closer estimation of the real transient power consumption, but it is also much slower as the scheduling of processes has to be calculated. Note that the power consumption estimated by this model will never be an upper bound of the power consumption, as it is defined by (4.2).

Table 4.1: Power values used in the example of the process-based power annotation model illustrated in Fig. 4.4.

Process	Core	Cache
A	10.0 mW	7.5 mW
B	5.0 mW	15.0 mW
C	2.5 mW	5.0 mW
D	15.0 mW	2.5 mW
E	12.5 mW	12.5 mW

4.2.3 Segment-Based Power Annotation Model

The process-based power annotation model already considers different power consumption characteristics of processes, but it does not distinguish between the various types of operations which might have different power consumption characteristics. Consider, for example, that not only the power consumption of the computation units affects the total power consumption, but

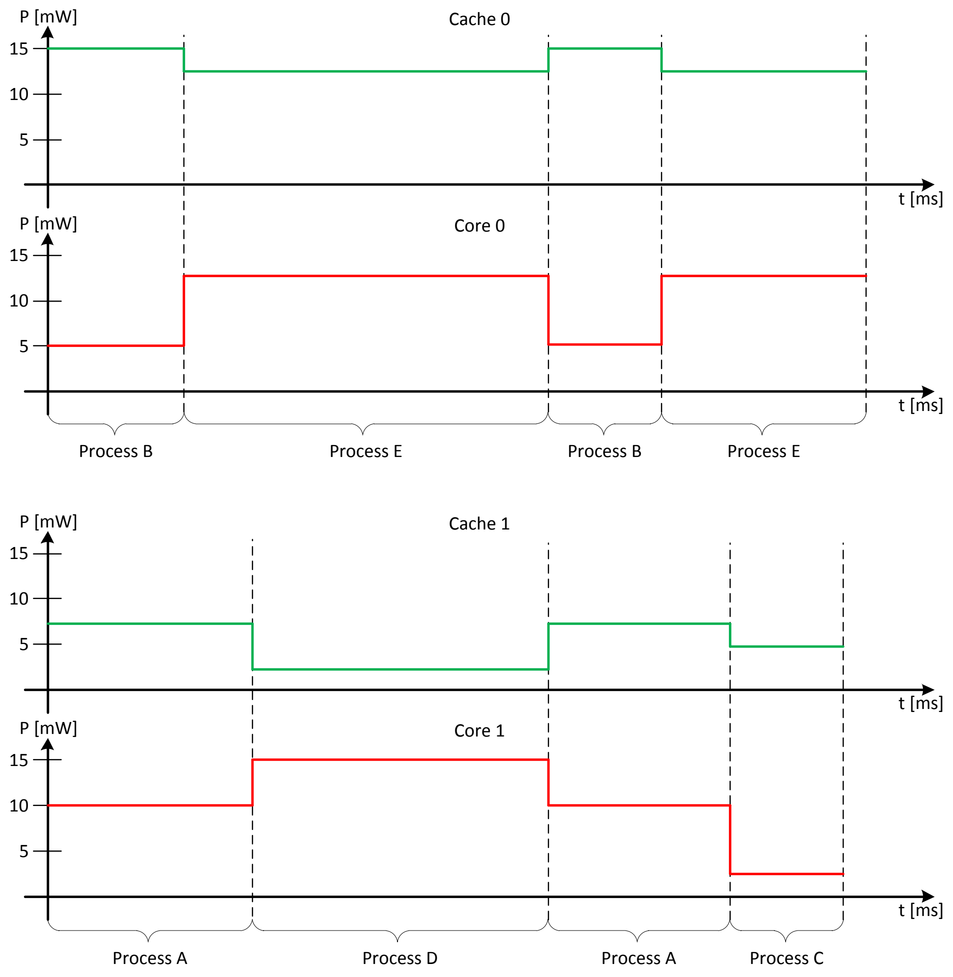


Figure 4.4: Example of the process-based power annotation model using a system with two homogenous tiles. The transient power consumption of a single process is constant.

also the one of the memories. In the prototype implementation of DOL for the MPARM virtual platform, the data scratchpad is only accessed during executing the `read` or `write` methods and therefore, the process only consumes dynamic power if it executes one of these methods.

In this subsection, the segment-based power annotation model is introduced. To address the above issues, it differentiates between various types of operations within a process. We use this model in SLTE to to estimate the power consumption. After introducing the notation of a segment, a brief overview of the power model is given. Afterwards, the application scheduling generation and power annotation are presented. Finally, the limitations of the model are listed and its value for the design space exploration is discussed.

Segments. The core idea of the segment-based power annotation model is to split a process in various segments of different types and individually annotate these segments with corresponding power values. Thus, before explaining the model in detail, we need to define a segment as well as the types of segments that are distinguished in this thesis.

A segment $s_{j,v} = \{i_{1,j}, i_{2,j}, \ldots, i_{k,j}\}$ is a logical grouping of consecutive instructions i within a process v and has the following properties:

- either no channel or exactly one channel is accessed during executing a segment,
- a segment consists of at least one instruction:

$$s_{j,v} = \{i_{1,j}, i_{2,j}, \ldots, i_{k,j}\}, \qquad k \geq 1 \tag{4.4}$$

- a segment that accesses a channel consists of exactly one instruction:

$$\exists i_{l,j} \in s_{j,v} \text{ access a channel } \Rightarrow s_{j,v} = \{i_{l,j}\} \tag{4.5}$$

- the collection of all segments $S_v = \{s_{1,v}, s_{2,v}, \ldots, s_{l,v}\}$ of a process v contains all instructions I_v of this process exactly once, thus S_v is an exact cover of I_v.

When comparing this definition of a segment with the definition of the KPN given in Section 4.2.1, three different types of segments can be identified:

- *Computing*: The segment consists of instructions that do not access a channel:

$$\begin{aligned}\mathfrak{T}(s_{j,v}) &= \text{Computing} \\ &\Leftrightarrow i_{m,j} \text{ does not access a channel } \forall m \in \{1 \ldots k\}\end{aligned} \tag{4.6}$$

- *Reading*: The segment only consists of a read instruction:

$$\mathfrak{T}(s_{j,v}) = \text{ Reading} \Leftrightarrow i_{1,j} \text{ is a read instruction} \tag{4.7}$$

- *Writing*: The segment only consists of a write instruction:

$$\mathfrak{T}(s_{j,v}) = \text{ Writing } \Leftrightarrow i_{1,j} \text{ is a write instruction} \tag{4.8}$$

Note that the process is only interacting with other processes in the *reading* or *writing* segments and therefore, a process only accesses the resources from a tile other than its own tile during executing segments of these types. To illustrate the principle of segments, the `fire` method of the simple process whose source code is outlined in Listing 4.1, is analyzed. The method reads

Listing 4.1: Source code sketch of the `fire` method of a typical process as used in a process network. In the method, a value is read from an input port, squared and finally written to the output port. Using the definition of a segment as given in Section 4.2.3, the method can be divided into five segments.

```
 1 int fire() {
 2     float i = 0;                    // computing, segment 1
 3     float j = 0;                    // computing, segment 1
 4 
 5     read((void*)PORT_IN, &i);       // reading,   segment 2
 6     j = i*i;                        // computing, segment 3
 7     write((void*)PORT_OUT, &j);     // writing,   segment 4
 8 
 9     return 0;                       // computing, segment 5
10 }
```

a value from an input port, squares it, and finally writes it to the output port. In this example, one can differentiate between five segments. The first segment, an *executing* segment, spans from Line 2 to Line 3. The second segment is of type *reading* and includes only `read` instruction on Line 5. The next segment includes Line 6, and therefore, it is of type *computing*, while the fourth segment is of type *writing* and contains the `write` instruction on Line 7. Finally, the last segment is the `return` statement on Line 9 and is of the type *computing*. As all of these five segments perform different operations, it is likely that all segments have a different power consumption.

Overview of the Model. After giving the definition of a segment, the segment-based power annotation model is briefly introduced next. In Fig. 4.5, a high-level overview of the model is given. The inputs to the segment-based power annotation model are the system specification, that is, the application, architecture, and mapping specification. Furthermore, the calibration data extracted from the low-level simulation is used to annotate the model with thermal and timing parameters. The *Application Schedule Creator* uses the system information and the timing parameters to compute a schedule of the system. Finally, in the *Power Annotation* block, the scheduling information is combined with the thermal parameters provided by the calibration data. The result of this block is the estimated transient power consumption of the system, also called power traces.

In Fig. 4.6, an example of the segment-based power annotation model is given. The system is similar to the one used in Section 4.2.2, but processes are separated in segments that have individual power consumptions.

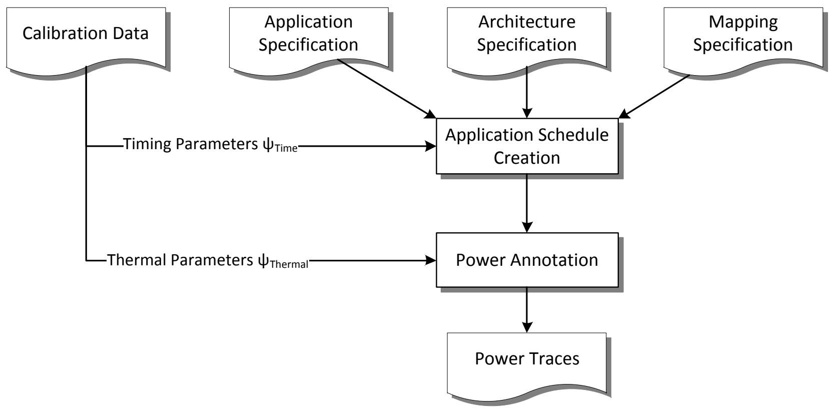

Calibration Data
Application Specification
Architecture Specification
Mapping Specification
Timing Parameters ψ_Time
Application Schedule Creation
Thermal Parameters ψ_Thermal
Power Annotation
Power Traces

Figure 4.5: High-level overview of the segment-based power annotation model.

Application Schedule. The transient power trace is created from two inputs, namely the thermal parameters and the scheduling information of the system. The scheduling information is represented as one timetable per tile that describes the temporal execution order of the processes bound to the corresponding tile. In the following, the *Application Schedule Creator*, whose output is the scheduling information, is discussed. The application scheduler of the process-based power annotation model is much simpler as the one introduced for this model, mainly as the knowledge of read and write events is missing.

Given a KPN $\mathcal{A} = (V, Q)$ as defined in Section 4.2.1, the timetable for tile γ is a sequence of segments θ_γ with the following properties:

- A timetable θ_γ contains only segments that are mapped to its tile γ:

$$s_{j,v} \in \theta_\gamma \Leftrightarrow b_v = \gamma. \tag{4.9}$$

.

- The timetable θ_γ considers all segments mapped to the tile γ:

$$\bigcup_{\forall \gamma \in \Gamma} \theta_\gamma = \bigcup_{\forall v \in V} \left(\bigcup_{\forall s \in S_v} s \right). \tag{4.10}$$

- The scheduling rules and policies of the modeled architecture are taken into account in the timetable.
- If no segment can be executed, the idle process P_{idle} is scheduled.

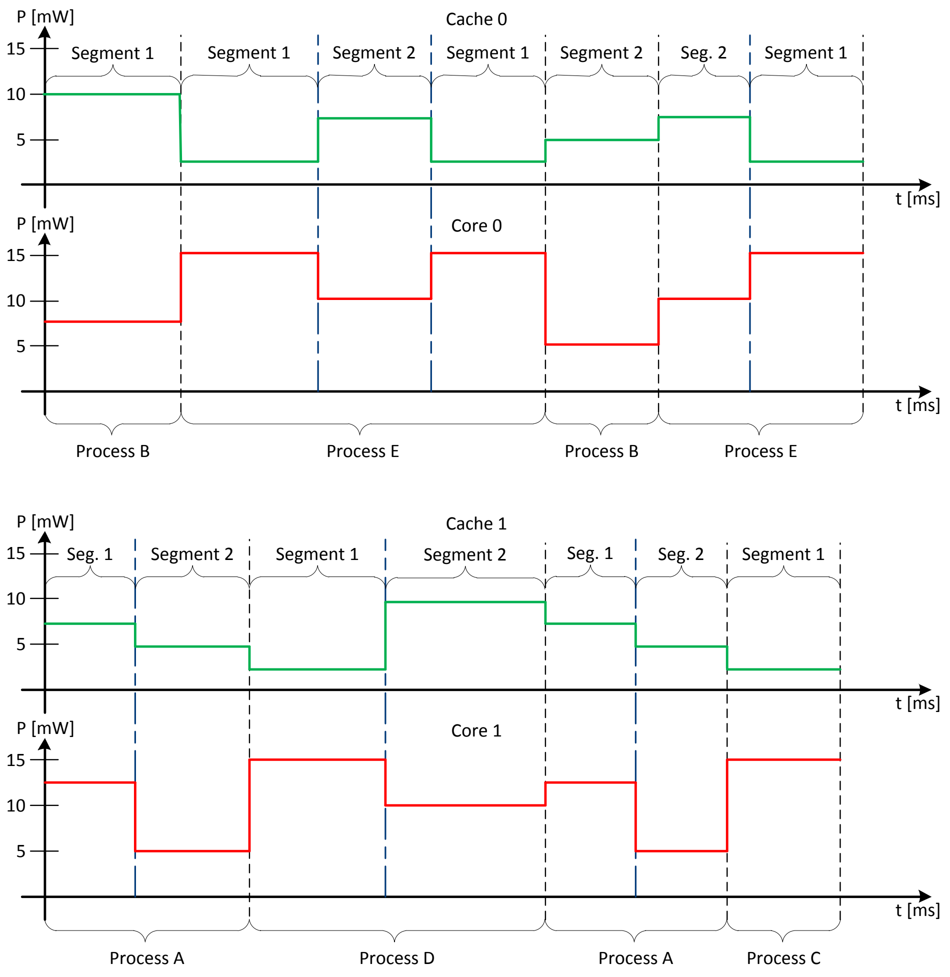


Figure 4.6: Example to illustrate the principle of the segment-based power annotation model using a system of two homogenous tiles.

In Table 4.2, an example of an application schedule is outlined. The architecture consists of totally three tiles, and each column contains the timetable of one single tile.

The level of abstraction used to create the scheduling information is a trade-off between accuracy and execution speed. An ISS provides high-accuracy, but is most probably too slow for design space exploration. A higher evaluation speed can be achieved by using a trace-based simulator. Recently, various implementations of a trace-based simulator for KPNs have been proposed in [64–66]. Nonetheless, their performance is often too slow for design space exploration and less accurate simulators are required. In [67], an Synchronous Dataflow (SDF) simulator is proposed to reduce the evaluation

Table 4.2: Example of an application schedule as it is created by the *Application Scheduler Creator*.

Time	Tile γ_1	Tile γ_2	Tile γ_3
5 ms	s_{1,p_2}	s_{1,p_1}	s_{1,p_4}
10 ms		s_{2,p_1}	
15 ms			s_{2,p_4}
20 ms	s_{1,p_3}		
25 ms		s_{1,p_5}	
30 ms	s_{2,p_3}	s_{2,p_5}	s_{3,p_4}
35 ms	Idle		Idle
40 ms			
45 ms	s_{2,p_3}	s_{3,p_1}	
50 ms	s_{2,p_2}		s_{3,p_4}

time even more. In the current version of our prototype implementation, an SDF simulator is implemented (see Section 6.2.1), as well.

Power Annotation. The *Application Schedule Creator* computes a timetable θ for each tile $\gamma \in \Gamma$. This scheduling information is forwarded to the power annotation block, where it is combined with the thermal parameters ψ_{Thermal} to estimate the transient power consumption of the system.

Given a system specification $S = \{\mathcal{A}, \mathcal{T}, (b, s)\}$ as defined in Section 4.2.1, the thermal parameters ψ_{Thermal} are defined as a mapping of a segment s to a collection of power values. The exact number of power values contained in this collection depends on the type of segment, but it always includes:

- A power value for each type of component c, which represents the power consumption of the component when the segment s is executed on the tile γ of this component.
- A power value for each type of component c_p, which might be involved in a communication step as a peer resource. This value represents the power consumption of the component when the segment s accesses the component c of tile γ_c from tile γ other than γ_c.
- A power value for each type of shared component sc, which is involved in the execution of the segment s.

We assume that the architecture consists of homogenous tiles, each with n components c, as it is the case for the target platform of this project, namely the MPARM virtual platform. Additionally, we assume that m components are involved as peer resources in the communication segments and that the

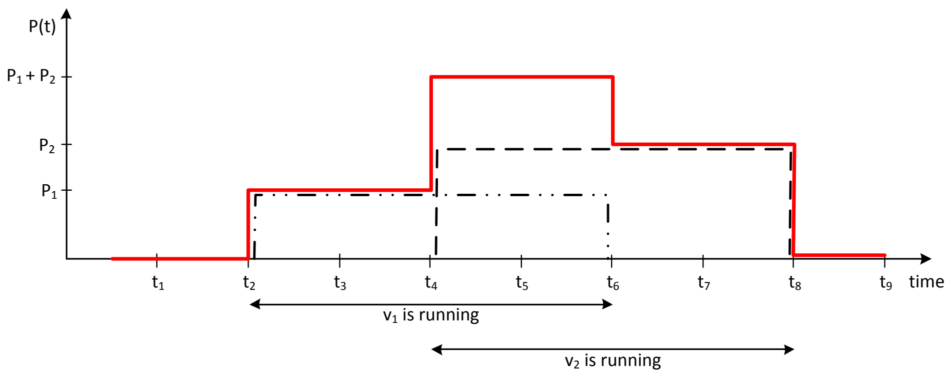


Figure 4.7: Example of power annotation when more than two processes concurrently access a shared resource.

architecture is composed of o shared components sc. Then, the thermal parameters of the calibration data can be expressed by

$$\psi_{\text{Thermal}} : s \mapsto \{c_1, \dots, c_n, c_{p,1}, \dots, c_{p,m}, sc_1, \dots, sc_o\}. \tag{4.11}$$

In heterogeneous systems, the definition of the thermal parameters remains the same with the only exception that more components have to be taken into account. Using the definition of the thermal parameters given in (4.11), the power annotation can be defined as

$$(\theta_\gamma, \psi_{\text{Thermal}}) \mapsto P_\gamma(t) \tag{4.12}$$

where $P_\gamma(t)$ is the transient power behavior of tile γ. Obviously, the power annotation of the components that are not shared between multiple tiles or involved in any communication transaction is a simple one-way assignment. However, the power annotation of all other components requires a slightly more difficult approach. Their total power $P_c(t)$ is defined as

$$P_c(t) = \sum_{s \in S_c} P_{s,c} \tag{4.13}$$

where S_c is the collection of all segments that might access the component c, either as a local component or as a peer component and $P_{s,c}$ is the power consumption of c when it is accessed by s.

The core idea of the power annotation of shared resources is outlined in Fig. 4.7. In this example, two processes, both running concurrently on different tiles, access the same shared resource. Process v_1 accesses it from time t_2 to time t_6 while process v_2 accesses the same resource from t_4 to t_8. This scenario corresponds for instance to the case when p_1 writes a huge amount of data to a scratchpad while process v_2 concurrently starts to read the data from the same scratchpad. From t_2 to t_6, the total power consumption is only the power consumption of process v_1 while between t_4 and t_6, the power consumption is the sum of the power consumption of both processes. Finally, between t_6 and t_8, the total power consumption corresponds to the power consumption of process v_2.

Limitations. Although the segment-based power annotation model considers various cases and its accuracy can be modularly adjusted by using a different scheduler, it has various limitations in performance and accuracy. In the following, the two core limitations of the model are discussed, that is, the constant power consumption and the lack of an upper bound of the power consumption.

The core idea of the power models discussed in this section is to abstract the power consumption of a component as a fixed number. Various granularities have been presented from an all-constant value to segment-based power consumption. Nonetheless, it is obvious that fixed power consumption might not lead to the real temperature. This is mainly important if one tries to use this model to estimate the peak temperature of a system. The use of the peak power consumption as power numbers would partly solve this problem, however, this may result in a very bad approximation of the real temperature curve as we will show in Chapter 7.

This leads to the second restriction: even though the peak power consumption of a segment is used as its power value, the segment-based power annotation model does not calculate an upper bound of the maximum temperature of a system. The *Application Scheduling Creator* assumes fixed execution times for the segments to create a timetable for each tile of the system. However, annotating these timetables with the worst-case power consumption of each segment might not lead to an upper-bound temperature as the following example shows. The application $\mathcal{A}$ consists of only one process p that has two segments $s_{1,p}$ and $s_{2,p}$. The execution times and the power consumptions of the processor of both segments are outlined in Table 4.3. One can easily verify that a higher temperature can be observed if segment $s_{1,p}$ only runs for 3 ms and $s_{2,p}$ runs for 4 ms for a given time span compared to the case where the execution time of both segments is always their average execution time. In reality, one has also to consider various other components like the memory or the cache, which might have a complete different power characterization with respect to the processor.

An approach to overcome this problem would be to calculate the segments that might run at a specific time and select the maximum power consumption of all these segments. However, the results form Yang et al. [68] show that such an approach is not feasible for design space exploration.

Table 4.3: Execution times and power consumptions of the processor used in an example application. BCET, ACET and WCET denote the best-case, the average-case and the worst-case execution time of the segment, respectively, and BCPC, ACPC, WCPC denote the best-case, average-case and worst-case power consumption of a segment, respectively. The discussed scenario in Section 4.2.2 is highlighted with a star.

	BCET	ACET	WCET	BCPC	ACPC	WCPC
$s_{1,p}$	3 ms$*$	4 ms	5 ms	8 mW	10 mW	12 mW
$s_{2,p}$	2 ms	3 ms	4 ms$*$	10 mW	12 mW	14 mW

Discussion. The abstract power model presented in this subsection is used to estimate the transient power consumption in SLTE, the proposed compositional temperature analysis model. By not only considering the computation, but also the communication aspects of a system, an accurate estimation of the transient power consumption can be calculated. Nonetheless, the model is limited in calculating an upper bound of the temperature as the computation of such a bound for an arbitrary streaming application executed on a distributed architecture can be very complex mainly for the following reasons:

- There is no direct correlation between the temperature of the system and the workload of the system. Furthermore, the temperature of a single component depends not only on the workload executed on this component, but also on the workload executed on all other components including the ones on all the other tiles.
- The transient power behavior is not directly correlated with the system specification as it depends on two characteristics: the power consumption itself and the scheduling information. Obviously, there is also no connection between the worst-case execution time and an upper bound of the transient power behavior.

In the second part of this thesis, we will revise this topic by proposing a framework to calculate the worst-case peak temperature of a many-core system.

Another point that needs to be discussed in this section is the performance. Self-evident, the segment-based power annotation model is much slower as the other power-annotation models presented in this section. The performance of the current model is mainly determined by the performance of the *Application Scheduling Creator*, thus, there exists a trade-off between performance and accuracy. In Chapter 6, we will limit the streaming application to the SDF model and improve the performance of the application scheduler creator by recognizing repetitive behavior in streaming applications.

4.3 Temperature Evaluation

A widely used duality to analyze the heat transfer of a modern VLSI system is to model the heat-flow as current passing through a thermal resistance and the thermal difference as the corresponding voltage [15, 51, 69, 70]. This duality is used to generate an *RC* system that describes the evolution of temperature when a specific power consumption is applied. Mathematically, this system can be represented by a first-order differential equation system and thus, the calculation of the temperature evolution is the most computationally intensive task of a thermal simulator.

As it is often too complicated to solve this differential equation system analytically, numeric approaches are favored. For example, in [17], a backward Euler method or in [14], an adaptive forth-order Runge-Kutta method is used to solve the differential equation system. In this section, we present an approach to improve the performance of the thermal simulation by partly precomputing the solution of the differential equation system. After introducing the basic thermal model, the proposed approach is presented in detail in Section 4.3.2. Finally, some advantages and drawbacks of the method are discussed in Section 4.3.3.

Note that a similar approach is used in [23, 40] for estimating the actual temperature of a system and then to reduce this temperature by adjusting the task scheduler of an OS. Compared to [23, 40], however, we use the method for improving the temperature behavior of an application offline, at system-level.

4.3.1 Thermal Differential Equation System

The temperature model used in this thesis is similar to the one introduced in [14]. In particular, the heat transfer can be expressed as a linear first-order differential equation:

$$\mathbf{C} \cdot \frac{\mathrm{d}\mathbf{T}(t)}{\mathrm{d}t} + \mathbf{G} \cdot \mathbf{T}(t) = \mathbf{P}(t), \qquad \mathbf{T}(0) = \mathbf{T}_0 \tag{4.14}$$

where $\mathbf{T}$ is the temperature vector, $\mathbf{C}$ a diagonal capacitance matrix, $\mathbf{G}$ the conductance matrix and $\mathbf{P}$ the power vector. All vectors have a dimension of $m \times 1$ and all matrices a dimension of $m \times m$. Both the capacitance and conductance matrix only depend on the architecture, that is, the floor-plan, but not on the input, namely the power consumption. Using $\mathbf{A} = \mathbf{C}^{-1} \cdot \mathbf{G}$ and $\mathbf{B} = \mathbf{C}^{-1}$, (4.14) can be rewritten as

$$\frac{\mathrm{d}\mathbf{T}(t)}{\mathrm{d}t} = -\mathbf{A} \cdot \mathbf{T}(t) + \mathbf{B} \cdot \mathbf{P}(t), \qquad \mathbf{T}(0) = \mathbf{T}_0. \tag{4.15}$$

4.3.2 Partly Precomputing the Solution of the Thermal Differential Equation System

Using the general solution of a non-homogeneous first-order differential equation system, the temperature can be calculated by

$$\mathbf{T}(t) = e^{-\mathbf{A}\cdot t} \cdot \mathbf{T}_0 + e^{-\mathbf{A}\cdot t} \int_0^t e^{\mathbf{A}\cdot\xi} \cdot \mathbf{B} \cdot \mathbf{P}(\xi)\,\mathrm{d}\xi. \tag{4.16}$$

To simplify this equation, the following assumptions are made:

1. During a specific time interval Δt, the power consumption is constant:
$$\mathbf{P}(t) = \mathbf{P} = const, \qquad 0 \leq t \leq \Delta t. \tag{4.17}$$
2. The time interval Δt is constant for the whole simulation:
$$\Delta t = const. \tag{4.18}$$
3. The only temperature of interest is at time Δt.

The first assumption follows from the fact, that a trace-based power measurement is considered. Thus, between two traces, the power consumption is fixed for all components. As the power is measured on a regular basis, the time interval between two traces is always constant and legitimates *Assumption 2*. As a thermal network has a pretty large time constant τ, the network behaves quite lazy The temperature does not change immediately, and it is enough to analyze the temperature at the end of a time interval Δt and *Assumption 3* is justified. Applying *Assumption 1* to (4.16), it leads to the following simplifications of the transient temperature:

$$\mathbf{T}(t) = e^{-\mathbf{A}\cdot t} \cdot \mathbf{T}_0 + e^{-\mathbf{A}\cdot t} \cdot \int_0^t e^{\mathbf{A}\cdot\xi}\,\mathrm{d}\xi \cdot \mathbf{B} \cdot \mathbf{P}, \qquad 0 \leq t \leq \Delta t. \tag{4.19}$$

Thus, the temperature can be expressed as

$$\mathbf{T}(t) = e^{-\mathbf{A}\cdot t} \cdot \mathbf{T}_0 + \mathbf{A}^{-1} \cdot \left(\mathbf{I} - e^{-\mathbf{A}\cdot t}\right) \cdot \mathbf{B} \cdot \mathbf{P}, \qquad 0 \leq t \leq \Delta t \tag{4.20}$$

with $\mathbf{I}$, the identity matrix of dimension $m \times m$. Using *Assumption 3*, the temperature at time Δt can be expressed as:

$$\mathbf{T}(\Delta t) = \underbrace{e^{-\mathbf{A}\cdot\Delta t}}_{\mathbf{E}} \cdot \mathbf{T}_0 + \underbrace{\mathbf{A}^{-1} \cdot \left(\mathbf{I} - e^{-\mathbf{A}\cdot\Delta t}\right) \cdot \mathbf{B}}_{\mathbf{F}} \cdot \mathbf{P} \tag{4.21}$$

where $\mathbf{E}$ and $\mathbf{F}$ are both independent of the input and mapping. As *Assumption 2* states that the time interval Δt is constant during the entire simulation, both matrices $\mathbf{E}$ and $\mathbf{F}$ can be calculated once during the design space exploration. By defining $\mathbf{T}[k] = \mathbf{T}(k \cdot \Delta t)$, the alteration of the temperature can be calculated by two matrix-vector multiplications:

$$\mathbf{T}[k+1] = \mathbf{E} \cdot \mathbf{T}[k] + \mathbf{F} \cdot \mathbf{P}[k]. \tag{4.22}$$

4.3.3 Discussion

Making assumptions always restricts the possible use cases of a method. In the following, the drawbacks introduced by our assumptions are discussed and the method is compared with the adaptive forth-order Runge-Kutta method.

The main drawback of the precomputed differential equation system method is the fixed step size that is required. Although this is the normal case when simulating a system on a low-level simulator, high-level simulators like the ones presented in Section 4.2, do not automatically calculate the power consumption on a regular basis. Nonetheless, it can be shown that a fixed step size in the high-level simulators introduces much less overhead as the use of a Runge-Kutta method to solve the differential equation system.

Another drawback of our method is that it can only be used if the system is linear, that is, the conductance matrix is constant. Our assumptions would become invalid in more complicated models with a temperature-dependent conductance matrix, as for example in a system with temperature-dependent thermal conductivity [71]. However, a linear approximation of the conductance matrix can be calculated by using Taylor's theorem to still apply the proposed approach in the design space exploration.

The most computationally intensive task of a thermal analysis model is the calculation of the temperature alteration. The presented method reduces the number of multiplications that need to be performed after every power trace by calculating the computationally intensive matrix exponential only once in the design space exploration. Therefore, the presented method is much faster than any numerical approach that solves the complete differential equation system after every trace. Furthermore, the method encourages the use of sparse matrix algorithms to improve the performance even further. Finally, note that the method does not reduce the accuracy of the solution as it is only a different approach to solve the thermal differential equation system.

4.4 Summary

The thermal evaluation is a key component in the design phase of a multiprocessor streaming application for both modern two- and three-dimensional systems. Violating thermal constraints of an IC may have serious consequences on the reliability of the system as well as on the ability to guarantee real-time constraints. To address thermal issues in an early design phase, fast and accurate thermal evaluation methods are required to estimate the thermal behavior of a candidate mapping already at system-level. In this chapter, SLTE, a system-level thermal evaluation model has been introduced, that is a first attempt to integrate compositional power evaluation into the design flow of multi-processor streaming applications.

However, including the thermal evaluation in the design space exploration can be a waste of time if all candidate mappings fulfill the thermal constrains of the architecture. Therefore, we have introduced a first very simple power analysis model that addresses the question if thermal analysis is even essential during design space exploration for a given multi-processor streaming application. Afterwards, a more refined, segment-based abstract power analysis model has been introduced and presented in detail. By considering running, reading, and writing segments separately, an accurate estimation of the temporal power consumption of an application can be computed. In the second part of the chapter, we have shown that the temperature alteration of a power trace can be calculated by two matrix-vector multiplications in the design space exploration. SLTE, our proposed compositional thermal evaluation model for the design space exploration, uses the segment-based abstract power evaluation model to calculate the power consumption of a candidate mapping. Afterwards, the temperature evolution of the system is calculated by the proposed method to partially precompute the solution of the thermal differential equation system.

In this chapter, we assumed that all required timing and thermal parameters are available and calculated in advanced. However, this data has to be extracted during model calibration. In Chapter 5, we will address this topic and propose various methods to generate the calibration data of a multi-processor streaming application using a low-level simulator.

5 Automated Model Calibration

Unlike low-level thermal evaluation methods, compositional thermal evaluation models use timing and thermal parameters to model the behavior of a multi-processor streaming application. In this context, model calibration refers to the extraction of the model parameters that are used to parameterize compositional thermal evaluation models. It is typically the first task that is performed during the design space exploration of a new multi-processor streaming application. Therefore, in this chapter, we will close the gap between the low-level simulation tool chain and the compositional thermal evaluation models by presenting an approach to use the thermal evaluation tool chain to automatically calibrate abstract evaluation models as, for example, the SLTE.

The presented approach includes the execution of a set of benchmark mappings in a low-level simulator to determine timing and power characteristics of the application. Afterwards, the thermal configuration of the architecture is extracted from a low-level thermal analysis model. This chapter is structured as follows: First, in Section 5.1, the required parameters and their sources are presented. In Sections 5.2 and 5.3, strategies to determine the timing and thermal parameters are discussed, respectively, and finally, a summary concludes the chapter.

5.1 Model Parameters and their Sources

As a large number of parameters from different sources is required to accurately characterize a multi-processor system, neither the task of extracting

Table 5.1: Overview of the timing and thermal parameters required by a compositional thermal evaluation model as, for example, the SLTE model.

Entity	Parameter	Unit	Source
segment s	average-/best-/worst-case execution time ACET(s), BCET(s), WCET(s)	sec / iteration	low-level sim.
	average / maximum average / peak power consumption	W	low-level sim.
queue q	minimal / maximal token size $N_{\min}(q), N_{\max}(q)$	bytes/access	functional sim.
	write rate, read rate $w(q), r(q)$	1	functional sim.
processor	clock frequency	cycles/sec	hardware data-sheet
architecture $\mathcal{T}$	capacitance matrix $\mathbf{C}$	J/K	low-level sim.
	conductivity matrix $\mathbf{G}$	W/K	low-level sim.

the timing and thermal parameters, nor the post-processing of the raw data should be underestimated. Considering the most common sources for model calibration, that is, physical implementations, simulation platforms, or formal methods, all of them have in common that various parameters cannot be directly extracted. As complex multi-processor streaming applications, like video decoders often require thousands of such parameters, manual model calibration is practically impossible.

In this section, we specify the timing and thermal parameters, and identify their sources. Unfortunately, we will show that there is not a single source where all parameters could be obtained from. In the following, the overall process of obtaining the timing and thermal parameters from a low-level simulation is described.

In Table 5.1, the most important parameters of a compositional thermal evaluation model are summarized. We will discuss these parameters in this and the following sections in detail. As already mentioned, the parameters can be categorized in two subsets. On the one hand, the timing parameters include the execution time and the characteristics of the channels, and on the other hand, the thermal parameters include the power consumption and the thermal configuration of the architecture. Although the categorization into timing and thermal parameters is the most obvious one, the parameters can also be categorized into their sources or in directly and indirectly observable parameters. While timing parameters are often directly measurable, the power consumption of a process is an example of an indirectly observable parameter, as additional post-processing is required to extract the individual power consumptions. In the context of this project, we differ between the following three sources to obtain the thermal and timing parameters:

- *Hardware data sheets*: The simplest sources for model parameters are the hardware data sheets. Compared to all other sources, it is the only source where parameters have to be obtained manually. Hardware data sheets are used to determine the clock frequency of microelectronic chips, and to extract various thermal parameters as, for example, the conductivity of the heat sink.
- *Functional simulation*: Mapping-independent parameters can mainly be extracted from the functional simulation. Examples of mapping-independent parameters are the minimal and maximal token size as well as the read and write rates of channels.
- *Low-level simulation*: A typical source to calibrate a system with their architectural and mapping-dependent parameters is a low-level virtual platform. The execution times and the power consumption can be extracted from a virtual platform, and the thermal configuration of the architecture is obtained from a low-level thermal analysis model.

5.2 Extracting Timing Parameters

The computation of an accurate scheduling of a multi-processor application requires various timing parameters. While the model calibration of an analytic analysis model has to calculate safe bounds on the characteristic parameters [72], abstract simulation models like SLTE require an average behavior of the timing parameters as the consideration of best-case and worst-case bounds makes it impossible to calculate the temperature evolution. In this section, we present methods to extract timing parameters: After introducing the idea of a functional simulator to extract mapping-independent parameters, the extraction of timing-dependent parameters using low-level simulation is illustrated. Again, we focus on applications modeled as KPN that are executed on distributed memory architectures.

The problem of obtaining timing parameters for performance models was already discussed in [29, 72] and our method is based on their approaches. However, the following points demonstrate the main differences between the two methods:

- The approach of Haid et al. [29, 72] mainly concentrates on extracting safe bounds of the timing parameters, while our method extracts the average behavior as well as bounds on the timing parameters.

- Instead of extracting all parameters process-based, all timing parameters are obtained separately for each segment.
- As the scheduling is only an interstage product to calculate the thermal behavior of a multi-processor streaming application, a way higher resolution is required in a thermal model as in a performance model. Therefore, the overhead of the OS, that is, the context switch and the process invocation, has to be determined, as well.

5.2.1 Functional Simulation

A functional simulator is an application that executes a given process network on a standard computer. One possibility to implement the process network is to map every process to a system thread. Obviously, this simulation is very fast, but can only be used to determine mapping-independent values. Possible parameters that can be determine by the functional simulation include the minimal and maximal token size $N_{\min}(q)$, $N_{\max}(q)$ and the read and write rate $r(q)$ and $w(q)$ of the queues. Practically, a monitor in the generic `read()` and `write()` methods of the functional simulator can be used to obtain these values. Note that the read and write rates are in general not constant for a KPN, and therefore, they need to be extracted separately for design alternative.

5.2.2 Low-Level Simulation

The source of all time dependent parameters of a multi-processor streaming application is a low-level simulation on a virtual platform. Cache misses, contentions on the bus or other features of modern processors cause these parameters to vary between every iteration and mapping. A viable strategy for the model calibration in the context of the design space exploration is to determine a safe bound and the average value of these timing parameters. For example, for the execution time, the Best Case Execution Time (BCET), ACET, and Worst Case Execution Time (WCET) are obtained and provided to the compositional thermal evaluation models. With T being the set of all measured values, the BCET can be expressed as

$$BCET = \min_{t \in T}(t), \tag{5.1}$$

the ACET as

$$ACET = \frac{\sum_{t \in T} t}{|T|}, \tag{5.2}$$

and the WCET as

$$WCET = \max_{t \in T}(t). \tag{5.3}$$

The execution times can be obtained in a similar manner to the functional simulation by monitoring the generic routines provided by the runtime environment. It is important to note that the observed quantities should not be affected by the monitoring itself. In a virtual platform, this can be guaranteed by using non-intrusive tracking methods provided by the platform itself.

Execution Times of the Segments. As mentioned in Section 4.2.3, three types of segments are distinguished: (1) *reading* from a FIFO channel, (2), *writing* to a FIFO channel, and (3) *computing*, that is, the process computes some calculations and is neither reading nor writing. As mentioned before, the execution time of these segments can be obtained by monitoring the generic functions of the runtime environment. In Listing 5.1, a typical process of a KPN is sketched. The method starts and stops with a *computing* segment and might be interrupted by some *reading* or *writing* segments. To determine the start and end time of an iteration, the runtime environment calls a monitor immediately before and after the fire method as sketched in Listing 5.2. The start and end points of a read and write segment are obtained by extending the corresponding stub methods of the runtime environment with monitor calls as outlined in Listing 5.3. We can show by induction, that this strategy can be used to differ between all *computing*,

Listing 5.1: Sketch of a typical process of a KPN. The `fire` method is called by the runtime environment in every iteration.

```
int fire() {
    // processing
    READ();
    // processing
    WRITE();
    // processing
}
```

Listing 5.2: Extension of the runtime environment to monitor the start and the end time of an iteration.

```
MONITOR(SEGMENT_START, COMPUTING, CURRENT_SIMULATION_TIME);
fire();
MONITOR(SEGMENT_STOP, COMPUTING, CURRENT_SIMULATION_TIME);
```

Listing 5.3: Extension of the read and write stubs of the runtime environment to monitor the start and end times of the read and write segments.

```
read() {
    MONITOR(SEGMENT_STOP, COMPUTING, CURRENT_SIMULATION_TIME);
    MONITOR(SEGMENT_START, READING, CURRENT_SIMULATION_TIME);
    // reading
    MONITOR(SEGMENT_STOP, READING, CURRENT_SIMULATION_TIME);
    MONITOR(SEGMENT_START, COMPUTING, CURRENT_SIMULATION_TIME);
}

write() {
    MONITOR(SEGMENT_STOP, COMPUTING, CURRENT_SIMULATION_TIME);
    MONITOR(SEGMENT_START, WRITING, CURRENT_SIMULATION_TIME);
    // writing
    MONITOR(SEGMENT_STOP, WRITING, CURRENT_SIMULATION_TIME);
    MONITOR(SEGMENT_START, COMPUTING, CURRENT_SIMULATION_TIME);
}
```

reading, and *writing* segments. Then, the execution times of a single segment can be calculated after the simulation using a parsing algorithm. Using an offline parsing algorithm, the execution times of the single segments can be calculated after the simulation.

Context Switch Overhead. In performance analysis, the overhead introduced by the context switch is often neglected in model calibration as its influence on the overall execution time is small. However, the power consumption of the context switch might be large as the context of the process is stored and restored from memory. Consequently, the extraction of its power consumption requires the recording of the temporal behavior of the context switch.

A simplified process state model with the three states *active*, *blocked*, and *ready* is outlined in Fig. 5.1. A process is *active* if it is currently executed, *ready* if it could be executed but another process is currently running and *blocked* if the process is waiting for some external events. Assuming that the scheduling policy is fixed priority, a necessary condition for a context switch is that *any* process is reading from or writing to a FIFO channel. Therefore, we can differ between two scenarios of a context switch:

1. A process is not able to read from or write to a FIFO channel and becomes blocked. The process notifies the OS, which in turn suspends the process, stores its context and restores the context of another process.
2. After being blocked, a process with a higher priority becomes able to read or write. The active process is interrupted and resumed by the OS. Afterwards, the OS stores the context of the currently running process and another process' context is restored.

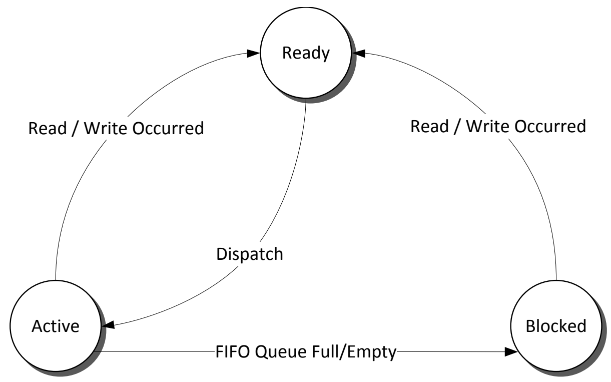


Figure 5.1: Simplified process state model to illustrate the context switch in a real-time OS under the assumption that the scheduling policy is fixed priority.

Monitors are added at the beginning and at the end of the suspending, resuming, context storing, and context restoring methods to extract the start and end times of the individual parts of the context switch. Note that the idle task is monitored, as well, to obtain the overhead introduced in switching the processor's state to idle and vice versa.

Runtime Environment Overhead. In addition to the context switch, the OS and the runtime environment of the process network generate some overhead in repetitively calling the `fire` method of the process. However, this overhead can easily be detected with the already existing monitors around the `fire` method.

5.3 Extracting Thermal Parameters

Obviously, a compositional thermal evaluation model requires a second category of parameters, that we call thermal parameters. They include the power consumption and the thermal configuration of the architecture. The thermal parameters are often obtained from two different sources: First, a two-step procedure is required to extract the power parameters with a low-level simulation platform. After measuring the transient power behavior as described in Section 3.2.3, the power traces are split into processes and segments, respectively, by using the timing information extracted by the methods introduced in the previous section. Secondly, the thermal configuration of the architecture is calculated by a low-level thermal analysis model.

The remainder of this section is organized as follows: First, some power metrics are briefly introduced while Section 5.3.2 presents the extraction of the actual power values from the power traces. The extraction of the power values from components that are used by more than one process in parallel is discussed in Section 5.3.3, and finally, in Section 5.3.4, the use of a low-level thermal analysis model for calculating the thermal characteristics of the architecture is presented in detail.

5.3.1 Power Metrics

Similar to abstracting the execution time of a process or segment as BCET, WCET or ACET, the power consumption is modeled by a power metric. In this project, three power metrics are used to express the power consumption of a segment or process: the average, maximum average, and the peak power consumption. The average power consumption is expressed as

$$P_{\text{avg}} = \frac{\sum_{i \in I} P_{\text{avg},i}}{|I|} \tag{5.4}$$

where I is the collection of all iterations and $P_{avg,i}$ is the average power consumption during iteration i. The second power metric, the maximum average power consumption is calculated by

$$P_{\text{max avg}} = \max_{i \in I} P_{\text{avg},i}. \tag{5.5}$$

Finally, the third power metric, the peak power consumption, is defined as

$$P_{\text{peak}} = \max_{i \in I} \left(\max_{t \in T_i} P(t) \right) \tag{5.6}$$

where T_i is the time span of iteration i. The determination of the average, maximum average, and peak power consumption is outlined in Fig. 5.2. Obviously, the peak power consumption P_{peak} of both iterations is the peak power of the *iteration* $i+1$, the average power consumption P_{avg} is the average of both average power consumptions of the *iteration* i and $i+1$ and the maximum average power consumption $P_{\text{max avg}}$ is the average power consumption of *iteration* $i+1$.

5.3.2 Extracting the Power Parameters

As already mentioned, the extraction of the power parameters from the power traces is performed in a two-step procedure:

1. The transient power behavior of each component is measured in a low-level simulation and stored as power traces.

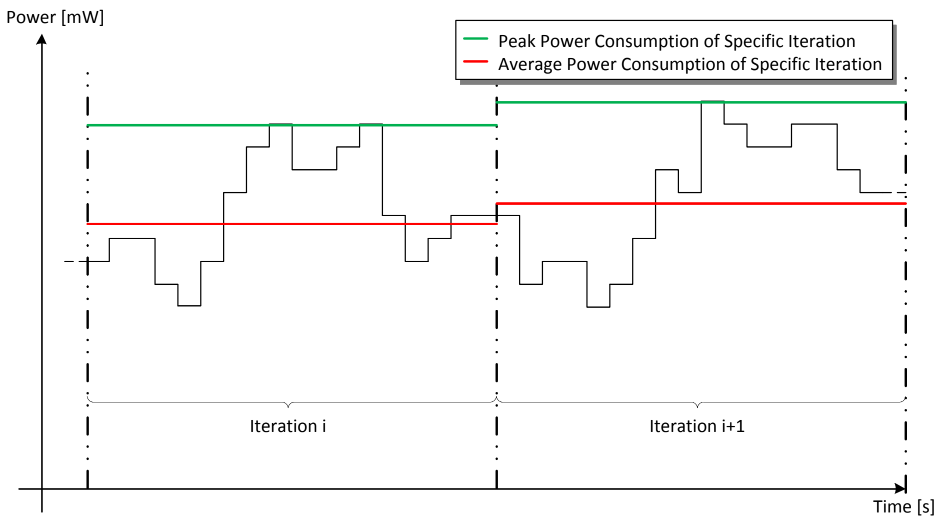


Figure 5.2: Example of the average and the peak power consumption over two iterations.

2. The power traces are segmented according to the desired granularity, that is, process-based or segment-based and any of the previous presented power metrics can be used to calculate the actual power values.

Note that is is not only calculated the power consumption of the individual segments and processes, but also the power consumed by the context switch and the runtime environment when completing and restarting an iteration.

5.3.3 Shared Elements

While the extraction of the power consumption is straightforward for components that are only used by one process per time, it is much more complicated for components that are used by multiple processes in parallel. The described problem is also illustrated in Fig. 5.3. Two processes, namely the producer and consumer process, are each mapped onto different tiles. It is assumed that the FIFO channel is implemented in the scratchpad of the producer process. As both processes are mapped onto a different tile, they can be executed in parallel and thus, can also simultaneously access the scratchpad of the producer. Note that simultaneously only means that two segments are running on parallel on the system level, which differs from the definition of simultaneously on the hardware level.

Figure 5.4 illustrates the power consumption extracted on a regular basis from a low-level simulation platform. As both the producer and the consumer process access the scratchpad in parallel, the total power consumption measured by the low-level simulation platform of the scratchpad is caused

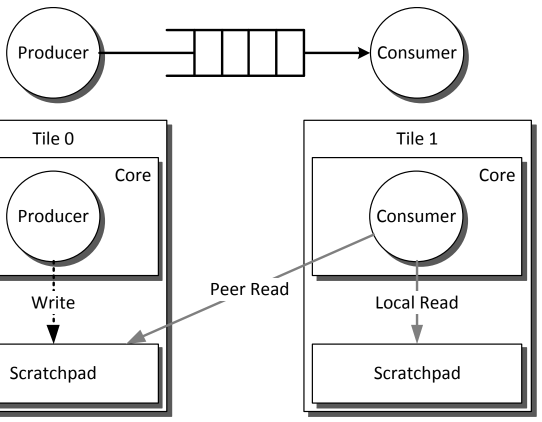


Figure 5.3: Process network to illustrate the simultaneously access of two processes to a shared resource.

by both processes and is averaged over the whole trace. However, a different candidate mapping might change the schedule in a way that both processes do no longer read or write in parallel. Therefore, total power consumption is divided into the power consumption caused by the producer process and the one caused by the consumer process. This approach is also illustrated in Figure 5.5.

In the general case, we have the following information available after a multi-processor streaming application has been executed on a low-level platform:

- The total average power consumption of a scratchpad per power trace,
- the starting and stopping points of all segments that read from and write to the scratchpad, and
- the power consumption ratio between a read and a write instruction of the scratchpad.

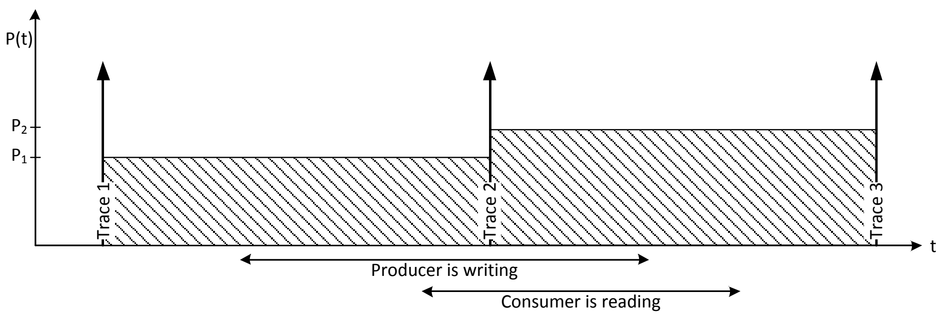


Figure 5.4: Measured power traces of the producer's scratchpad of the example outlined in Fig. 5.3. Both the producer and the consumer process access the scratchpad in parallel.

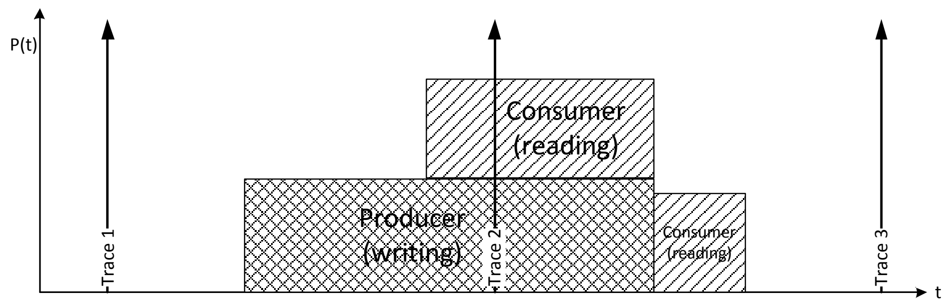


Figure 5.5: Actual power consumption of the example outlined in Fig. 5.3. The total power consumption is divided into the power consumption caused by the producer process as well as the one caused by the consumer process.

The latter depends on the implementation of the channels in the runtime environment and can be calculated by using the hardware data sheets of the scratchpad and by considering the implementation of the FIFO channels, that is, the read and write stubs of the runtime environment. In this subsection, we address the problem of separating the total power consumption of a shared resource into the individual processes that access the resource in parallel.

The basic idea is to split the total energy consumption, that is, the area below the power curve, among the individual processes. Afterwards, the average power consumption can be calculated by dividing the individual energy consumptions by the amount of time that the process accessed the shared resource. The total energy consumption in trace $tr_i \in Tr$ can be computed by

$$E_{tr_i} = \Delta P_{tr_i} \cdot t_{tr_i} = \sum_{v \in V} t_{v,tr_i} \cdot \Delta P_{v,tr_i} \tag{5.7}$$

where E_{tr_i} is the total energy consumed in trace tr_i, ΔP_{tr_i} is the average power consumption in trace tr_i and t_{tr_i} is the length of trace tr_i. Furthermore, $v \in V$ is the set of processes accessing the scratchpad in trace tr_i, t_{v,tr_i} the amount of time that process v is executed in trace tr_i and $\Delta P_{v,tr_i}$ the average power consumption of process v in trace tr_i. As all processes are either reading from or writing to the scratchpad, the power consumption can be recapped as

$$P_{reading} = P_v \quad \forall v \in V_{reading} \tag{5.8}$$

and

$$P_{writing} = P_v \quad \forall v \in V_{writing} \tag{5.9}$$

where $V_{reading}$ is the set of all processes that read from the scratchpad and $V_{writing}$ is the set of all processes that write to the scratchpad. Using (5.8) and (5.9), (5.7) can be rewritten as follows:

$$\begin{aligned}\Delta P_{tr_i} \cdot t_{tr_i} = &\sum_{v \in V_{reading}} t_{v,tr_i} \cdot \Delta P_{reading} \\ &+ \sum_{v \in V_{writing}} t_{v,tr_i} \cdot \Delta P_{writing}.\end{aligned} \tag{5.10}$$

The power consumption ratio r of a read and a write instruction

$$r = \frac{P_{writing}}{P_{reading}} \tag{5.11}$$

is known in advance. Expressions (5.10) and (5.11) form a system of two equations, that can be solved for the power consumption $P_{reading}$ of a reading process and the power consumption $P_{writing}$ of a writing process. Algorithm 5.1 shows the pseudocode to extract the power consumption from the total power consumption of a component that is simultaneously used by multiple processes. However, as a process always copies the data from the peer scratchpad to its local scratchpad, the algorithm differs between accesses to the local and peer scratchpad, as well.

Note that the algorithm presented in this subsection is for communication channels that are implemented in the scratchpad of the producer or consumer. However, only minimal modifications to the algorithm are necessary for using it with other software channel communication implementations like shared memories.

5.3.4 Temperature Parameters

Compositional thermal evaluation models use timing and power parameters to model the power consumption of a multi-processor streaming application. Additionally, the thermal configuration of the architecture is required to model the temperature behavior of the application. The thermal configuration includes the capacitance matrix $\mathbf{C}$, the conductivity matrix $\mathbf{G}$, and the ambient temperature T_0. These parameters can simply be extracted from a low-level thermal analysis model that uses the floor-plan and the thermal configuration as input, and creates the thermal differential equation system.

As several compositional thermal evaluation models require different parameters for the thermal configuration, additional post-processing is required. A numerical computing environment is often most suitable for this task as it includes multi-dimensional computations like calculation of the matrix exponential.

Algorithm 5.1: Pseudo-code to extract the power consumption of an individual process from the total power consumption of a component that is simultaneously used by multiple processes. The local tile denotes the tile that the process is mapped onto while the peer tile is the tile of the peer process.

```
procedure PowerExtraction()
    for all trace $tr \in Tr$ do
        instantiate a collection $M$ per tile $\gamma \in \Gamma$
        for all segment $s \in S$ do                  ▷ find all active segments in $tr$
            if $s$ is a read or write segment and $s$ is active in $tr$ then
                $t \leftarrow$ amount of time that $s$ was active in $tr$
                $M$(local tile).put($s$, type←local, $t$)  ▷ influence local scratchpad
                $M$(peer tile).put($s$, type←peer, $t$)   ▷ influence peer scratchpad
            end if
        end for
        for all tile $\gamma \in \Gamma$ do        ▷ calculate individual power consumption
            $E_{tr} \leftarrow$ energy in trace $tr$                 ▷ total energy in trace $tr$
            $w \leftarrow$ TotalWeight($S$)
            for all segment $s$, time $t_s$, $type \in M(\gamma)$ do
                $f \leftarrow t_s *$ SegmentFactor($s$, $type$) / $w$
                $E_s \leftarrow f \cdot E_{tr}$                    ▷ energy fraction of segment $s$
                $P_s \leftarrow E_s / t_s$                          ▷ power of segment $s$
            end for
        end for
    end for
end procedure

function TotalWeight(Segments $S$)
    $w = 0$
    for all segment $s$, time $t_s$, $type \in S$ do
        $w \leftarrow w + t_s *$ SegmentFactor($s$, $type$)
    end for
    return $w$
end function

function SegmentFactor(Segment $s$, $type$)
    if $s$ is a write segment then
        return writefactor($type$)                       ▷ ratio when writing
    else if $s$ is a read segment then
        return readfactor($type$)                        ▷ factor when reading
    end if
end function
```

5.4 Summary

Compositional thermal evaluation models like SLTE outperform low-level simulation in the design space exploration by providing a much faster evaluation time of the candidate mappings. However, compositional thermal evaluation models require timing and thermal parameters to model the behavior of a multi-processor streaming application. As the modeling of the thermal behavior often requires thousands of parameters, manual model calibration becomes practically impossible. Talking this challenge, we presented strategies to automatically calibrate a compositional thermal evaluation model.

Unfortunately, there does not exist a single source for all model parameters. Hardware data sheets, functional, and low-level simulations are required to extract all kind of parameters. Clock frequencies and the thermal characteristics of the heat sink are extracted by considering hardware data sheets. The read and write rate of the channels are examples of parameters determined with a functional simulation as they are often independent of the architecture and mapping. Finally, low-level simulation is used to determine bounds on the execution time and the power consumption as well as to calculate the thermal characteristics of the architecture.

The execution times of processes and segments are modeled by their BCET, ACET, and WCET. Similarly, we defined three power metrics to model their power consumption. After introducing various approaches to extract the power consumption of a segment or a process from a low-level simulation platform, the extraction of the thermal characteristics of the application from a low-level thermal analysis model has been discussed. In summary, this chapter has enabled the use of a compositional thermal evaluation model in the design space exploration by providing approaches to automatically calibrate such models with the use of a low-level thermal evaluation tool chain.

6 System-Level Thermal Simulation in DOL

In the previous chapters, the core ideas of a fast and accurate compositional thermal evaluation model have been described. To demonstrate the viability of our approaches, we present the details of a prototype implementation of SLTE in DOL in this chapter. To calibrate the model with the timing and thermal characteristics of a multi-processor streaming application, a set of benchmark mappings has been executed on a low-level simulation platform. Although using the MPARM virtual platform and the HotSpot analysis model for the model calibration, porting the presented prototype implementation to another platform is relatively easy as the low-level simulators are only used for the model calibration.

This chapter is structured as follows: First, the extension of a low-level tool chain consisting of the MPARM simulator and the HotSpot thermal analysis model for the automated model calibration is discussed. In Sections 6.2 and 6.3, a prototype implementation of SLTE is presented. After discussing the computation of the power consumption in DOL in Section 6.2, its extension with the precomputed thermal analysis method is presented in Section 6.3. Finally, a summary concludes the chapter.

6.1 Automated Model Calibration

The automated model calibration of SLTE is performed by executing a set of benchmark mappings on a low-level simulation platform. First, the thermal evaluation tool chain presented in Chapter 3 is extended to measure the execution times and the power consumption of the application. Afterwards, a post-processing method is used to calculate the timing and power parameters. Finally, the thermal characteristics of the architecture are extracted from the low-level thermal analysis model, that is, HotSpot, and is post-processed to be used in SLTE. In the following, a prototype implementation of each of these steps is illustrated.

6.1.1 Simulation and Measurement

Two different simulators are used for the extraction of the timing and thermal parameters. In the following, both simulators are shortly introduced.

Functional Simulation. The functional simulation is used to obtain the mapping-independent parameters. An already existing functional simulator described in [29, 64] is used for this purpose. As the simulator is based on the SystemC libraries, it uses SystemC threads for the computational tasks and SystemC channels to implement the FIFO queues. As no timing-information is extracted from the functional simulator, a data-driven scheduler is used to control the application.

Low-Level Simulation. The architecture and mapping-dependent parameters are obtained from a low-level simulation. The thermal evaluation tool chain introduced in Chapter 3 is used to measure the execution times and the power consumption of the multi-processor streaming application. Although both types of data are obtained from the same simulation, different methods are used to extract the timing behavior and the power consumption. On the one hand, to extract the timing information of an application, it is monitored by macros included in the runtime environment. On the other hand, the power consumption of the application is measured as power traces similar to the approach presented in Section 3.2.3. As the simulator samples the power consumption on a regular basis, there is no need to extend the runtime environment for the measurement of the power dissipation.

The workflow of the thermal evaluation tool chain is as follows: Based on the system specification, DOL generates the required glue code for executing a multi-processor streaming application on the distributed architecture. This glue code, together with the source code of the streaming application, is compiled to run on top of the RTEMS OS. Finally, the binaries can be executed in the MPARM simulator, and the execution times and the power consumption can be measured. Mainly two functionalities of the glue code have to be extended for the model calibration: The process wrappers with the ability to measure the start and end points of an iteration and the communication primitives with the ability to recording read and write activities.

Algorithm 6.1: Code snippet of the glue code that invokes the actual process by calling the `fire` method. Monitors are added immediately before and after the process invocation to record the start and end time of the process. The `log` method prints the current simulation time together with the event type.

```
while (true) do
    log("invocation_start")        ▷ log the start of the process invocation
    fire()                         ▷ invoke the process
    log("invocation_end")          ▷ log the end of the process invocation
end while
```

In Algorithm 6.1 outlines a code snipped of the extended glue code to record the start and end time of the process invocation. Immediately before and after the process invocation, the current simulation time and the event type, that is, invocation start or invocation end, are logged. A similar approach is also used to extend the communication primitives with the ability to record the start and end time of a FIFO channel invocation. Algorithm 6.2 outlines a code snipped of the write method that enables the measurement of the execution times of all segments according to their definition in Section 4.2.3. Note that the `log` method calls a script that is recognized by the simulator for recording, and therefore, its influence on the execution time is negligible. All logged events are written to a logging file that can later be post-processed to analyze the detailed system behavior.

The same technique is also used to measure the characteristics of the context switch as well as the one of the idle task. However, to monitor the context switch, additional logging asserts are required. As described in Section 5.2.2, two scenarios for a context switch can be distinguished: (1) a process becomes blocked and therefore, the OS suspends the process or the (2) the OS resumes a new process after an interrupt. Therefore, five different events are distinguished to record the characteristics of the context switch. We refer to Table 6.1 for a detailed description of all events. The same asserts are used to monitor the characteristics of the idle task, that is, to record the

Algorithm 6.2: Code snippet of the `DOL_Write` method as included in the glue code for the MPARM platform. The `log` method prints the current simulation time together with the event type.

```
procedure DOL_Write(size,buffer)
    log("write_start")             ▷ log the start of the write method
    write(size, buffer)            ▷ write the data to the FIFO
    log("write_end")               ▷ log the end of the write method
end procedure
```

Table 6.1: Event types to monitor the context switching of the RTEMS OS. Whenever multiple event names are listed for the same description, they all denote the same event.

Event	**Description**
Start Suspend	Process is unable to read (FIFO channels are empty) or write (FIFO channels are full). Therefore, the process becomes blocked and the OS suspends the process.
Start Interrupt	An interrupt occurs as a process has read from or has written to a FIFO channel that is used by a blocked process bound to this processor.
End Suspend End Interrupt Start Store	The context switch handler starts to store the context of the currently running process.
End Store Start Restore	The context switch handler starts to restore the context of the new process.
End Restore	The context of the new process is completely restored and the execution of the new process can continue.

overhead of switching the processor's state to idle and vice versa. The later is necessary, as these stages often have much higher power consumptions as the idle process itself.

6.1.2 Timing and Power Parameter Extraction

In the last subsection, the thermal evaluation tool chain has been extended to log the start and end times of segments and to monitor the characteristics of the context switch. Furthermore, the temporal power characteristics of a component are measured by power traces. However, additional post-processing is required to obtain the model parameters. In the following, a prototype implementation of a three-step procedure to calculate the timing and thermal parameters is presented.

Following the workflow outlined in Fig. 6.1, the multi-processor streaming application is simulated in the thermal evaluation tool chain that monitors the start and end points of the segments and calculates the temporal power consumption. In the first step, the *MPARM Logger* parses the log file to extract the running times of the segments and to calculate their average execution time. The second and third step are responsible to back-annotate the timing and the power parameters to an extended process network file. Besides the

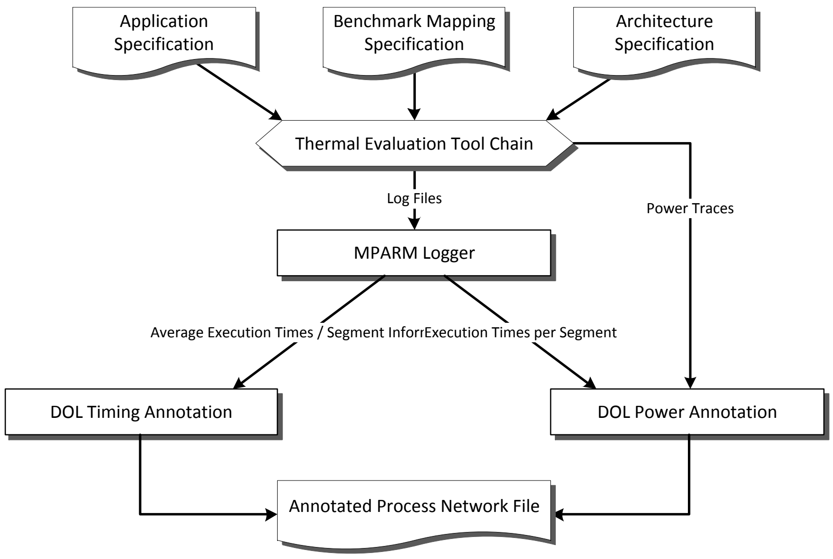

Application Specification
Benchmark Mapping Specification
Architecture Specification
Thermal Evaluation Tool Chain
Log Files
Power Traces
MPARM Logger
Average Execution Times / Segment InforrExecution Times per Segment
DOL Timing Annotation
DOL Power Annotation
Annotated Process Network File

Figure 6.1: Three-step procedure to calculate the timing and thermal parameters from the output of the thermal evaluation tool chain.

original information about the process network, it contains the profiling data of the application. In Listing 6.1, an example of such an extended process network file is given. While the only task of the *Timing Annotation* block is to write the average execution times to the extended process network file (e.g., Lines 13 and 28 in Listing 6.1), the *Power Annotation* block calculates the power parameters and writes them to the extended process network file (e.g., Lines 19 to 21 in Listing 6.1). To extract the power consumption of the context switch, its timing behavior is split into various segments, as well. As all these context switch segments are only used whenever the execution flow requires it, they are stored as subelements of the segment that was executed at the time of the context switch (e.g., Lines 14 to 17 in Listing 6.1).

6.1.3 Extracting the Thermal Configuration

To take advantage of the characteristics of the design space exploration to reduce the evaluation time of the thermal analysis, the precomputed thermal analysis model, which has been introduced in Section 4.3, is used to calculate the temperature in SLTE. Besides the temporal power consumption, the thermal analysis model requires two matrices, that characterize the thermal differential equation system. These matrices are extracted in a two-step procedure from the thermal evaluation tool chain.

In the first step, the low-level thermal analysis model, that is, HotSpot [14], is extended to write the capacitance matrix **C**, and the conductivity matrix

Listing 6.1: Extended XML specification of a process network as used to store the thermal and timing parameters in the prototype implementation of SLTE. A comma-separated format is used to store the power consumption of the individual components. ACPWR denotes the power consumption calculated with the average power consumption metric, MACPWR the one calculated with the maximum average power consumption metric and finally, PEAKPWR denotes the power consumption calculated with the peak power consumption metric.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<processnetwork>
    <process name="generator" basename="generator" range="">
        <port name="1" basename="1" range="" type="output" />
        <source type="c" location="generator.c" />
        <profiling name="NumOfFires" value="60" />
        <profiling name="1.accesses" value="1" />
        <profiling name="1.tokensize" value="4" />
        <profiling name="1.initialAccesses" value="0" />
        <profiling name="1.initialtokensize" value="0" />
        <profiling name="NumOfSegments" value="4" />
        <profiling name="seg.0.TYPE" value="COMPUTING" />
        <profiling name="seg.0.ACET" value="1036" />
        <profiling name="seg.0.INTERRUPT" value="2705" />
        <profiling name="seg.0.RESTORE" value="926" />
        <profiling name="seg.0.SUSPEND" value="2279" />
        <profiling name="seg.0.STORE" value="570" />
        ...
        <profiling name="seg.0.MACPWR" value="
            46.9600,63.8083,8.8267,0.0000,24.1899,0.3336,0.0126" />
        <profiling name="seg.0.ACPWR" value="
            45.7866,53.9803,7.3715,0.0000,15.0240,0.2465,0.0093" />
        <profiling name="seg.0.PEAKPWR" value="
            46.9600,63.8083,8.8267,0.0000,26.3541,0.3336,0.0126" />
        <profiling name="seg.0.INTERRUPT.MACPWR" value="
            44.4932,41.5235,7.4479,0.0000,20.1316,0.1815,0.0068" />
        <profiling name="seg.0.INTERRUPT.ACPWR" value="
            44.4932,41.5235,7.4479,0.0000,20.1316,0.1815,0.0068" />
        <profiling name="seg.0.INTERRUPT.PEAKPWR" value="
            46.2009,57.6663,8.3935,0.0000,23.5495,0.1868,0.0070" />
        ...
    </process>
    ...
    <profiling name="idle.STORE.ACET" value="609" />
    ...
    <profiling name="static" value="
        1.50,3.11,2.00,14.63,1.25,4.05,4.05,0.00,0.00,0.79,0.31,0.45"
        />
    <profiling name="idle.STORE.MACPWR" value="
        46.37,60.14,8.09,0.00,11.05" />
    ...
</processnetwork>
```

$\mathbf{G}$ to a text file. In the second step, a Matlab script is generated and executed that calculates the following characteristic matrices:

$$\mathbf{E} = e^{-\mathbf{C}^{-1} \cdot \mathbf{G} \cdot \Delta t} \tag{6.1}$$

and

$$F = \left(\mathbf{C}^{-1} \cdot \mathbf{G}\right)^{-1} \cdot \left(\mathbf{I} - e^{-\mathbf{C}^{-1} \cdot \mathbf{G} \cdot t}\right) \cdot \mathbf{C}^{-1}. \tag{6.2}$$

To not overload the extended process network file, that is used to store the thermal and timing parameters, both matrices are stored in separate text files.

6.2 Calculating the Temporal Power Consumption

To demonstrate the viability of the proposed approaches of a compositional thermal evaluation model, we present the details of a prototype implementation of SLTE. Before demonstrating the extension of DOL with the precomputed thermal analysis model in the next section, an implementation of the segment-based power annotation model is presented in the following. Note that all other abstract power models presented in Section 4.2.2 are simplifications of the segment-based power annotation model and therefore, a corresponding prototype implementation can easily be derived from the one presented in this section. Following the structure of the segment-based power annotation model, the scheduling creation is first presented in Section 6.2.1. Afterwards, in Section 6.2.2, the power annotation step is illustrated in detail.

6.2.1 Scheduling Creation

As the computation of the timetables takes the most time to calculate the power consumption of a design alternative, a fast scheduling creator is essential in the design space exploration. In Section 4.2.3, the segment-based power annotation model has been described for KPNs. However, KPN applications have various drawbacks in the area of performance analysis: The data dependency of the process behavior makes the calculation of a static schedule impossible and arbitrary complex control flows are possible [65]. Therefore, we restrict ourselves to applications modeled as an SDF [73, 74]. In the following, another scheduling creator for SDF applications is presented that guarantees both adequate accuracy and a high speed-up, and is included in the proposed prototype implementation of SLTE.

The restriction to SDF applications for performance analysis is quite common in research [29, 75, 76], mainly due to fact that the control flow can be calculated in advance. However, as most multi-processor streaming applications' origin is the field of signal processing, they can easily be modeled as an SDF. An SDF is defined as a "network of synchronous blocks" [73] and can formally be described as a graph $\mathcal{A} = (V, Q, W)$. The definitions of the nodes V and the edges Q correspond to the ones of a KPN as given in Section 4.2.1. $W = \{w_1, \ldots, w_{|Q|}\}$ describes the token consumption behavior of an SDF and is the main restriction of an application modeled as an SDF compared to the KPN model. The token consumption w_q of the channel q is represented as a triple (p_e, c_e, d_e). Assuming q connects the node v_1 and v_2, the number of tokens produced by node v_1 in every execution is denoted as p_e. The second parameter of the triple, c_e is the number of tokens consumed by v_2 in every execution and the initial amount of token in the channel q is called d_e as it represents the delay.

An SDF simulator has been developed for the computation of the scheduling of the application with respect to the new mapping. The simulator models the behavior of the RTEMS OS and the MPARM platform by creating an application flow sequence of the segments that conforms to the new mapping. A segment is characterized by its average execution time and an adequate accuracy is guaranteed by modeling the interruption of processes and the storing and restoring of process states according to the description given in Section 6.1.1.

In Algorithm 6.3, the process scheduling algorithm of the SDF simulator is outlined in pseudocode. Fixed priority is assumed as scheduling policy in the algorithm and there exists an instance of this scheduling algorithm for each tile $\gamma \in \Gamma$. The algorithm is executed whenever the state of a process bind to tile γ is changed. By introducing the subsegments `SUSPEND`, `RESUME`, `STORE`, and `RESTORE`, the algorithm takes into account the overhead caused by the context switch. Obviously, these subsegments are only scheduled whenever a new process becomes the non-blocked process with the highest priority. After restoring the context of a new process, it might happen that a process with a higher priority is not blocked any more. Therefore, the infinite loop (Line 2) guarantees that a process could directly be interrupted after being restored.

In the design space exploration, the time to evaluate a design alternative is critical. The consideration of the repetitive behavior of an SDF application can significantly reduce this amount of time. Due to the facts that (1) the FIFO channels only have a limited token size, (2) the scheduling policy is fixed priority, and (3) and all segments are modeled by constant execution times, the sequence of segments is continuously repeated. In Fig. 6.2, a simple process network is outlined. Both the producer and the consumer process are mapped onto different tiles that are connected by a FIFO channel with a capacity of three tokens. After startup, the pattern of executing one producer iteration and one consumer iteration in parallel is continuously

Algorithm 6.3: Pseudocode of the process scheduling algorithm when fixed priority is selected as scheduling policy in the SDF scheduler. There exists an instance of this code for each tile $\gamma \in \Gamma$.

```
Require: $V_s = \{v_1, \ldots, v_n\}$ the set of all processes mapped to this processor.
    The processes are sorted by descending priority, that is, $\mathcal{P}\{v_i\} \geq \mathcal{P}\{v_{i+1}\}$
    with $\mathcal{P}\{v\}$ the priority of $v$.
Require: $a$, the current active process
 1: procedure NextProcessToSchedule()
 2:   loop
 3:     switch state $s$
 4:     case: NONE
 5:       if all processes are done then
 6:         $a \leftarrow$ IdleTask
 7:         return DONE
 8:       end if
 9:       $v \leftarrow$ first process of $V_s$ that is not blocked.
10:       if all processes are blocked then
11:         $a \leftarrow$ IdleTask
12:         return STALLING
13:       else if new process is already active then
14:         return COMPUTING
15:       else
16:         if $a$ is STALLING then
17:           schedule *suspend* of $a$, $s \leftarrow$ SUSPEND
18:         else
19:           schedule *resume* of $a$, $s \leftarrow$ RESUME
20:         end if
21:         return COMPUTING
22:       end if
23:     case: RESUME
24:     case: SUSPEND
25:       schedule *store* of $a$, $s \leftarrow$ STORE
26:       return COMPUTING
27:     case: STORE
28:       schedule *restore* of $v$, $s \leftarrow$ RESTORE
29:       return COMPUTING
30:     case: RESTORE
31:       $a \leftarrow v$, $s \leftarrow$ NONE
32:       continue
33:     end switch
34:   end loop
35: end procedure
```

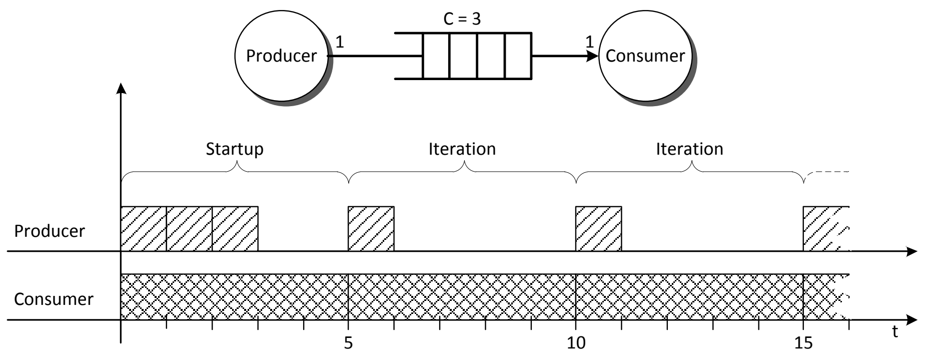


Figure 6.2: Example of the repetitive behavior of an application modeled as an SDF. The processes have a fixed execution time and are connected by a FIFO channel with a capacity of three tokens.

repeated. Obviously, such an application can be in one of the three stages that are summarized in Table 6.2. Note that an application might not enter the repetitive stage if the capacity of the FIFO channels is too big and the number of iterations is too small. In the current version of the prototype implementation of the SDF simulator, we have implemented a detection mechanism for the repetitive stage so that the creation of their timetables only have to be performed once.

Table 6.2: Overview of the different stages that an application, modeled as an SDF, can adapt in the SDF simulator.

Startup stage	Stage until all FIFO channels reach their steady-state behavior.
Repetitive stage	Stage where a pattern of segments is repetitively executed.
End stage	Stage where no other iteration of the repetitive stage can be executed until the first process is completed and the time period until all other processes are completed.

6.2.2 Power Annotation

In the second part of the prototype implementation of the segment-based power annotation model, the timetables created by the SDF simulator are combined with the thermal parameters, that is, the power consumption of the segments, to estimate the transient power behavior of the system. In this subsection, the modular structure of the power annotator used in the prototype implementation is shortly described.

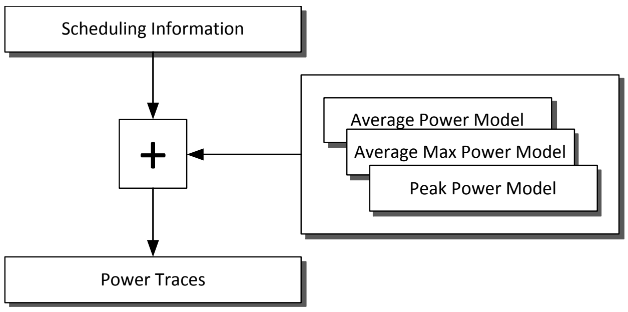


Figure 6.3: Sketch of the power annotator as used in the prototype implementation of SLTE.

A rough sketch of the power annotator implementation is outlined in Fig. 6.3. In our framework, we have implemented three power models designed according to the power metrics introduced in Section 5.3.1. Each of the models, the average power model, maximum average power model, and peak power model, is annotated with the power values calculated by the corresponding power metric formulas. The power annotator contains one package for each power model; however, due to the modular design of our implementation, it could easily be extended by new power models.

The composition of the power values and the scheduling information differs between components that are only used by one process per time (e.g., the core or the cache) and components that are concurrently used by all processes (e.g. scratchpads or shared memories). The power consumption of the components that are only used by one process per time corresponds to the power values of the segment that is currently executed. However, the power consumptions of the other components are the sum of the power values of all segments that access the component at this time.

6.3 System-Level Thermal Simulation in DOL

In SLTE, the precomputed thermal analysis model introduced in Section 4.3 is used to calculate the temperature evaluation of a design alternative and to reduce the computational effort to two matrix-vector multiplications per power trace. In the following, the implementation of the precomputed thermal analysis model in DOL is briefly introduced. Afterwards, based on the thermal model of HotSpot, a simple approach to reduce the number of multiplications by another 25 % is illustrated.

Motivated by the fact that the thermal analysis model has only to calculate two matrix-vector multiplications, we implemented the model as an extension of DOL. However, as DOL is written in a high-level programming language, that is, Java[1], additional native libraries are required to perform the matrix-vector multiplications in an efficient manner. For the prototype implementation of SLTE, we use Matrix-Toolkits-Java (MTJ)[2] to compute the linear algebra operations. MTJ is based on the netlib-java library[3] that can be configured to use a native optimized implementation of BLAS[4] and LAPACK[5] for efficient matrix-vector multiplications.

Four layers are used to model a 2D processor in HotSpot: the silicon, the interface, the heat spreader and the heat sink layer. Assuming a floor-plan with n components, the total number of nodes in the model is $m = 4 \cdot n + 12$, where the factor 4 accounts the four layers and the additional summand models extra nodes for the spreader, the heat sink and the peripheral heat sink [69]. Using the precomputed thermal analysis model, the number of multiplications per power trace can be reduced to $2 \cdot m^2$, which is independent of the selected interval length or accuracy.

From the definition of the thermal model, it follows that each node has an associated temperature value. Furthermore, every node of the silicon layer has the corresponding power dissipation as power value, and both the nodes of the heat sink layer and the extra nodes are used to model the influence of the ambient temperature. As the nodes of the interface and the heat spreader layer have no associated power values, we can reduce the number of multiplications by almost another forth as calculated next:

$$
\begin{aligned}
\mathbf{F} \cdot \mathbf{P} &= \begin{pmatrix} f_{11} & \cdots & f_{1m} \\ \vdots & \ddots & \vdots \\ f_{m1} & \cdots & f_{mm} \end{pmatrix} \cdot \begin{pmatrix} p_1 \\ \vdots \\ p_n \\ 0 \\ \vdots \\ 0 \\ p_{3 \cdot n+1} \\ p_m \end{pmatrix} \\
&= \begin{pmatrix} f_{11} & \cdots & f_{1n} & f_{1\{3 \cdot n+1\}} & \cdots & f_{1m} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ f_{m1} & \cdots & f_{mn} & f_{m\{3 \cdot n+1\}} & \cdots & f_{mm} \end{pmatrix} \cdot \begin{pmatrix} p_1 \\ \vdots \\ p_n \\ p_{3 \cdot n+1} \\ p_m \end{pmatrix}
\end{aligned}
\tag{6.3}
$$

[1]http://www.java.com/

[2]http://code.google.com/p/matrix-toolkits-java/

[3]http://code.google.com/p/netlib-java/

[4]http://www.netlib.org/blas/

[5]http://www.netlib.org/lapack/

6.4 Summary

The evaluation of potential design alternatives in the design space exploration of a multi-processor streaming application requires fast and accurate methods for performance and thermal analysis. Low-level simulators like MPARM [13] provide very detailed information about the power and timing behavior of an application, however, because they are too expensive in terms of execution time, they seem to be inadequate for design space exploration.

In this chapter, specific details of a prototype implementation of SLTE have been illustrated to demonstrate the viability of the concepts introduced in Chapters 4 and 5. The thermal evaluation tool chain has been extended with various post-processing methods to automatically calibrate SLTE with its timing and thermal parameters. Having various advantages for thermal evaluation, we use the SDF model as design methodology of our prototype implementation. To reduce the evaluation time of a candidate mapping in the design space exploration, an SDF simulator has been developed that recognizes the repetitive behavior of a multi-processor streaming application. In the next chapter, the discussed implementation of SLTE is used to demonstrate the viability of the proposed approaches by analyzing the thermal behavior of three multiprocessor streaming applications.

7 Experimental Results

In the first part of this thesis, SLTE, a compositional temperature evaluation model for multi-processor streaming applications, has been proposed as an efficient method for the thermal evaluation of design alternatives. In this chapter, three case studies are used to highlight the key benefits and demonstrate the viability of the proposed method. A prototype implementation of the thermal evaluation tool chain is used for the automated model calibration. Furthermore, the previously discussed extension of DOL with SLTE performs the thermal evaluation of the design alternatives. A producer-consumer application, a distributed matrix multiplication application, and a Motion JPEG (MJPEG) application are used to analyze the proposed model in detail.

After describing the experimental setup in Section 7.1, the mentioned applications are introduced and various measurements are reported in Section 7.2. Afterwards, Section 7.3 discusses the accuracy and the speed-up of the results and finally, a summary concludes the chapter.

7.1 Experimental Setup

The MPARM virtual platform and the HotSpot thermal analysis model introduced in Chapter 3 have been used for the automated model calibration and to compare the outcomes of the abstract models. In the following, the hardware/software platform, the thermal setup, the experimental setup, and the evaluation metric are described.

Hardware/Software Multi-Processor Platform. The MPARM platform [13] presented in Section 3.2 is used as hardware platform to run all applications. The ARM7 cores and the shared bus operate at a clock frequency of 200 MHz. The platform has the advantage that its number of tiles can easily be adjusted. The RTEMS OS [48] that was shortly introduced in Section 3.2.1, has the ability to provide the functionality of quasi-parallelism and has an already existing FIFO implementation. In all experiments, RTEMS is used as real-time OS and the scheduling policy is selected to be fixed priority. Compared to the version of RTEMS provided by the MPARM platform, the implementation of the idle task has been adjusted. Originally implemented as a busy wait, the idle task switches the processor state of the ARM7 processor to the idle mode [77] to reduce the power consumption in the prototype implementation.

Power and Thermal Setup. The power model of the MPARM simulator introduced in Section 3.2.2 is used to calculate the power consumption in all experiments. The power consumption of the MPARM platform is multiplied by a factor of five for a better illustration of its impact. Note that the absolute power value has no influence on the viability of the proposed approach as the power model of SLTE is based on the thermal parameters extracted from the MPARM simulator. Therefore, both the system-level and the low-level simulation models use the same power values. In case the power model of MPARM does not support the memory sizes of our architecture, the power values of MPARM are interpolated to the required size by taking advantage of CACTI [78]. In all experiments, a sampling interval of 100 μs is used to generate the power traces.

The floor-plan of this prototype implementation is based on the examples given in [35], but they are adjusted to the MPARM platform and the number of tiles of the concrete architecture. The thermal configuration that is used in [79] for the HotSpot model, simulates almost no heatsink. Therefore, we adapted it for our prototype implementation, as well. In Table 7.1, the detailed thermal configuration used in all experiments is summarized. Again note that the viability of the model is independent of the thermal configuration as all thermal parameters are extracted from HotSpot.

Experimental Setup. All experiments are performed on a Intel Core 2 Duo E6600 with 2 GByte RAM and `GCC` version 3.4.3 is used as compiler for all simulators and applications. The prototype implementation of DOL is compiled with `Java` version 6.

Evaluation Metric. To determine the performance and the accuracy of our proposed approaches, we simulate all applications in the thermal evaluation tool chain introduced in Chapter 3. In the following, speaking from

Table 7.1: Thermal configuration of HotSpot as used in this project. Most of the values are borrowed from [79] to neglect the effects of the heatsink.

Parameter	Symbol	Value
Silicon thermal conductivity [W/(m · K)]	k_{chip}	100
Silicon specific heat [J/(m^3 · K)]	p_{chip}	$1.75 \cdot 10^6$
Thickness of the chip [mm]	t_{chip}	0.8
Convection resistance [K/W]	r_{convec}	13.9
Heatsink width [m]	s_{sink}	0.011
Heatsink thickness [mm]	t_{sink}	0.1
Heatsink thermal conductivity [W/(m · K)]	k_{sink}	400
Heatsink specific heat [J/(m^3 · K)]	p_{sink}	$3.55 \cdot 10^6$
Thickness of the interface material [mm]	$t_{interface}$	0.025
Ambient temperature [K]	T_{amb}	300

low-level simulation always refers to the thermal evaluation tool chain. The speed-up denotes the ratio between the execution time of the low-level simulator and the prototype implementation of SLTE. The execution time of the low-level simulation corresponds to the sum of the execution time of the MPARM simulator and the one of the HotSpot thermal analysis model.

To quantify the distance between the maximum temperature of the low-level simulation and the prototype implementation of SLTE, we introduce the *pessimism* metric. It measures their normalized difference, i.e. normalized by the maximum variation of the temperature, and is defined as

$$pessimism = \frac{|T^* - \max(T_{sim})|}{margin} \tag{7.1}$$

where the *margin* is the thermal difference between the maximum temperature when executing the system with utilization 100% and 0%, T^* is the maximum temperature of the prototype implementation of SLTE and $\max(T_{sim})$ is the maximum temperature occurred in the low-level simulation of the application. The pessimism is calculated separately for each component and an average of all pessimisms is used as metric of the accuracy.

7.2 Applications

The viability of the proposed model is analyzed with three multi-processor streaming applications. The synthetic producer-consumer application is used to illustrate the behavior of SLTE for different lengths of process chains and observe the impact of the segment's execution time on the accuracy of SLTE.

The scalability of the proposed model is studied with the matrix application where the multiplication of two matrices is distributed over many processes. Finally, the MJPEG decoder is used to confirm the observed trends with a real-world application.

The calibration of the thermal model follows the procedure described in Chapter 5. Unless stated otherwise, the same mapping is used for model calibration and thermal analysis. However, if various candidate mappings are compared, a single, randomly selected mapping is used for calibration[1]. Note that the following characteristics are outlined for each evaluation scenario (Tables 7.2 to 7.8):

1. the average, maximum, and the minimum pessimism when the average power consumption is used as power parameter for SLTE,
2. the actual execution time of the application on the real platform, that is, the MPARM platform, and
3. the time that is used to simulate the system in SLTE and the corresponding speed-up of SLTE compared to the low-level simulation.

A detail description of all mappings used in this section and supplementary results of the applications as well as a Fast Fourier Transform (FFT) and Infinite Impulse Response (IIR) application are listed in Appendix A.

7.2.1 Producer-Consumer

The producer-consumer application is a simple synthetic application that consists of three or more processes that are mapped on a distributed architecture with three processors. Its process network is outlined in Fig. 7.1 for a chain of five processes. The functionality of the application can be described as follows: The *producer* process generates a stream of floating point numbers which are passed to the first *worker* process. Every *worker* process squares the floating point number various times before passing the value to the next *worker* process. Finally, the last *consumer* process prints the received value. To compare the results among themselves, the ability to recognize the repetitive behavior of the application has been deactivated in all experiments.

Three different scenarios are investigated with the producer-consumer application. First, the execution time of an iteration of the *worker* process is altered. The considered structure of the application consists of three *worker*

[1] In the following, the mapping that is used to perform the model calibration is marked with a star.

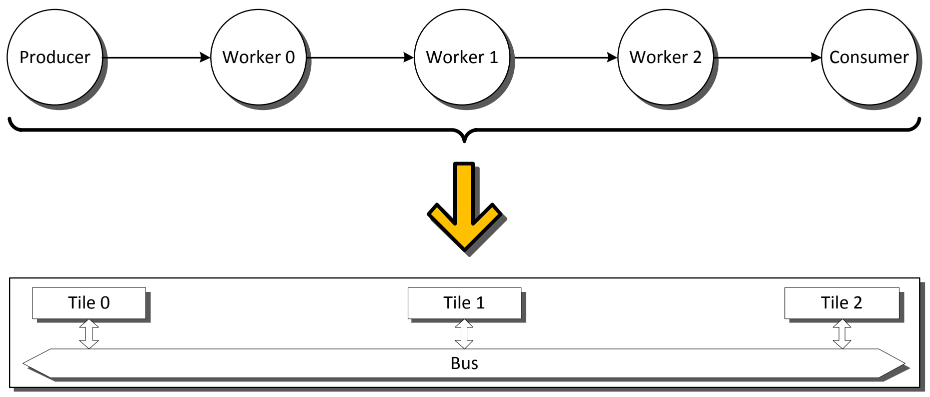


Figure 7.1: Process network of the producer-consumer application with three *worker* processes. The application is mapped onto a distributed architecture of three tiles.

processes and the corresponding results are outlined in Table 7.2. In the second scenario, the number of *worker* processes is altered from one to eight processes. Again, the pessimism, the execution times and the speed-up are summarized in Table 7.3. Finally, in the third scenario, seven different mappings of the producer-consumer application are compared. The application contains three *worker* processes and its characteristics are reported in Table 7.4.

Table 7.2: Producer-Consumer, Execution Time of Process:
Pessimism, execution times and speed-ups of the first evaluation scenario. The execution time T of the *worker* process is altered. The chain is extended to three *worker* processes and mapped onto a distributed architecture of three tiles. The same mapping is used for model calibration and evaluation.

	Pessimism			**Execution Time**		
	Average	*Min*	*Max*	*Platform*	*SLTE*	*Speed-up*
T = **0.2** ms	0.0036	0.0017	0.0084	2.45 s	2.20 s	1678
T = **0.4** ms	0.0021	0.0015	0.0075	2.37 s	1.73 s	2000
T = **1.0** ms	0.0011	0.0007	0.0026	2.33 s	1.63 s	2046
T = **2.0** ms	0.0006	0.0001	0.0027	2.31 s	1.52 s	2146
T = **4.0** ms	0.0009	0.0003	0.0047	2.30 s	1.45 s	2230
T = **10** ms	0.0005	0.0000	0.0040	2.30 s	1.32 s	2457
T = **20** ms	0.0003	0.0000	0.0028	2.29 s	1.24 s	2559
T = **40** ms	0.0003	0.0000	0.0034	2.29 s	1.30 s	2676

Table 7.3: Producer-Consumer, Process Chain Length:
Pessimism, execution times and speed-ups of the second evaluation scenario. The number of *worker* processes (WPs) is altered from one to eight processes, but the application is always mapped onto a distributed architecture of three tiles. The same mapping is used for model calibration and evaluation.

	Pessimism			**Execution Time**		
	Average	*Min*	*Max*	*Platform*	*SLTE*	*Speed-up*
1 WP	0.0032	0.0001	0.0090	2.37 s	1.71 s	2013
2 WPs	0.0015	0.0003	0.0021	2.46 s	1.89 s	2108
3 WPs	0.0019	0.0005	0.0024	2.48 s	1.79 s	2245
4 WPs	0.0013	0.0008	0.0030	2.50 s	1.77 s	2560
5 WPs	0.0014	0.0010	0.0033	2.26 s	1.70 s	2214
6 WPs	0.0010	0.0004	0.0024	2.27 s	1.69 s	2416
7 WPs	0.0010	0.0008	0.0021	2.43 s	1.74 s	2386
8 WPs	0.0009	0.0002	0.0036	2.44 s	1.77 s	2473

Table 7.4: Producer-Consumer, Different Mappings:
Pessimism, execution times and speed-ups of the third evaluation scenario where seven different mappings are compared. The chain is extended to three *worker* processes and is mapped onto a distributed architecture of three tiles. The mapping that is used to perform the model calibration is marked with a star.

	Pessimism			**Execution Time**		
	Average	*Min*	*Max*	*Platform*	*SLTE*	*Speed-up*
Mapping 1$*$	0.0019	0.0005	0.0024	2.48 s	1.78 s	2244
Mapping 2	0.0033	0.0014	0.0040	2.48 s	1.78 s	2243
Mapping 3	0.0048	0.0022	0.0053	1.34 s	1.58 s	1806
Mapping 4	0.0022	0.0013	0.0028	2.48 s	1.77 s	2282
Mapping 5	0.0035	0.0000	0.0092	3.72 s	2.10 s	2488
Mapping 6	0.0039	0.0006	0.0051	3.73 s	2.13 s	2459
Mapping 7	0.0021	0.0005	0.0034	2.43 s	1.75 s	2251

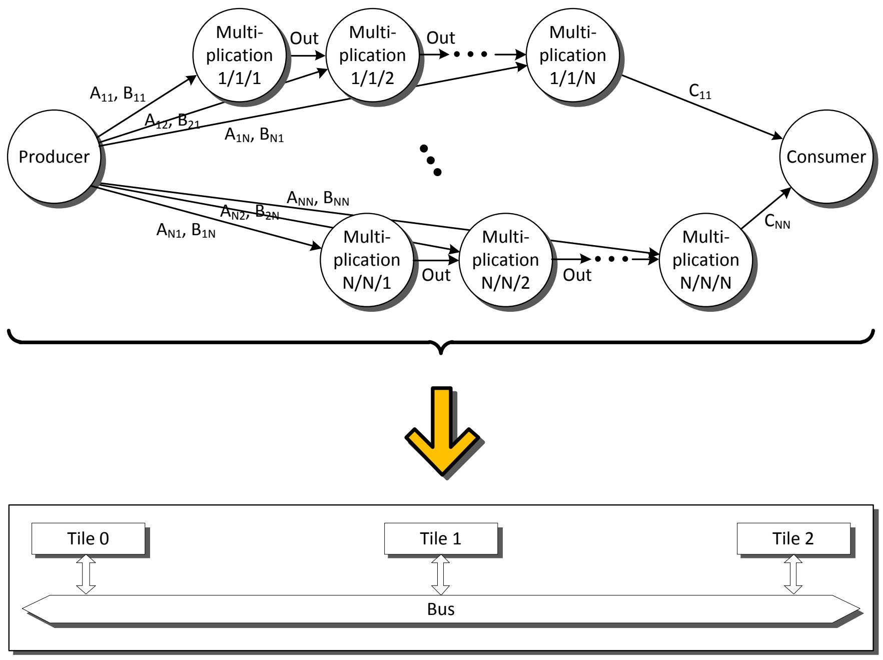


Figure 7.2: Process network of a distributed implementation of the matrix multiplication. Every multiplication process executes exactly one multiplication and addition. The application is mapped onto a distributed architecture of three tiles.

7.2.2 Matrix Multiplication

The matrix multiplication application calculates an $N \times N$ matrix multiplication in a distributed way by splitting up the matrix product into single multiplications and additions. Therefore, the resulting streaming application outlined in Fig. 7.2 consists of N^3 multiplications, an input generator, and an output consumer process. Assuming the application is used to calculate the product C of the $N \times N$ matrices A and B. The input to the multiplication process $(i/j/k)$ is A_{ij}, B_{ji} and the output of the multiplication process $(i/j/k-1)$. It calculates $A_{ij} \cdot B_{ji} + \mathrm{out}_{(i/j/k-1)}$ and forwards this to the multiplication process $(i/j/k+1)$. The first process of a line, that is, $(i/j/1)$ has zero as input instead of the previous result and the last process of a line, that is, $(i/j/N)$ forwards the result to the output consumer process. The application is mapped onto a distributed architecture of three tiles.

The matrix multiplication application is used to investigate two different evaluation scenarios. In the first scenario, the viability of SLTE is compared for three different dimensions of the matrix, namely $N = 1$, $N = 2$, and

Table 7.5: Matrix Multiplication, Dimension:
Pessimism, execution times and speed-ups of the first evaluation scenario where three different matrix dimensions N are compared. The application is mapped onto a distributed architecture of three tiles and the same mapping is used for model calibration and evaluation.

	Pessimism			**Execution Time**		
	Average	*Min*	*Max*	*Platform*	*SLTE*	*Speed-up*
$\mathbf{N = 1}$	0.0030	0.0021	0.0087	2.61 s	1.57 s	2775
$\mathbf{N = 2}$	0.0010	0.0003	0.0027	2.52 s	1.81 s	2195
$\mathbf{N = 3}$	0.0018	0.0001	0.0042	2.52 s	2.32 s	1760

Table 7.6: Matrix Multiplication, Different Mappings:
Pessimism, execution times and speed-ups of the second evaluation scenario where seven different mappings are compared. The application is mapped onto a distributed architecture of three tiles and the mapping that is used to perform the model calibration is marked with a star.

	Pessimism			**Execution Time**		
	Average	*Min*	*Max*	*Platform*	*SLTE*	*Speed-up*
Mapping 1$*$	0.0010	0.0003	0.0027	2.52 s	1.81 s	2195
Mapping 2	0.0017	0.0001	0.0046	2.31 s	1.57 s	2413
Mapping 3	0.0076	0.0058	0.0113	2.59 s	1.64 s	2487
Mapping 4	0.0012	0.0002	0.0022	2.32 s	1.49 s	2571
Mapping 5	0.0014	0.0004	0.0020	2.32 s	1.49 s	2565
Mapping 6	0.0025	0.0008	0.0070	2.32 s	1.48 s	2574
Mapping 7	0.0055	0.0040	0.0083	2.70 s	1.73 s	2420

$N = 3$. The pessimism, the execution times and the speed-ups are reported in Table 7.5. Various mappings are the topic of the second evaluation scenario and its results are summarized in Table 7.6.

In Fig. 7.3, the temporal temperature evolution of the first mapping of Table 7.6 is plotted. Other than the temperature recorded from the low-level simulation, the temperature simulated with SLTE is plotted for three different power annotations, that is, the average, maximum average, and peak power consumption introduced in Section 5.3.1. Similarly, the temporal temperature evolution of the seventh mapping is plotted in Fig. 7.4.

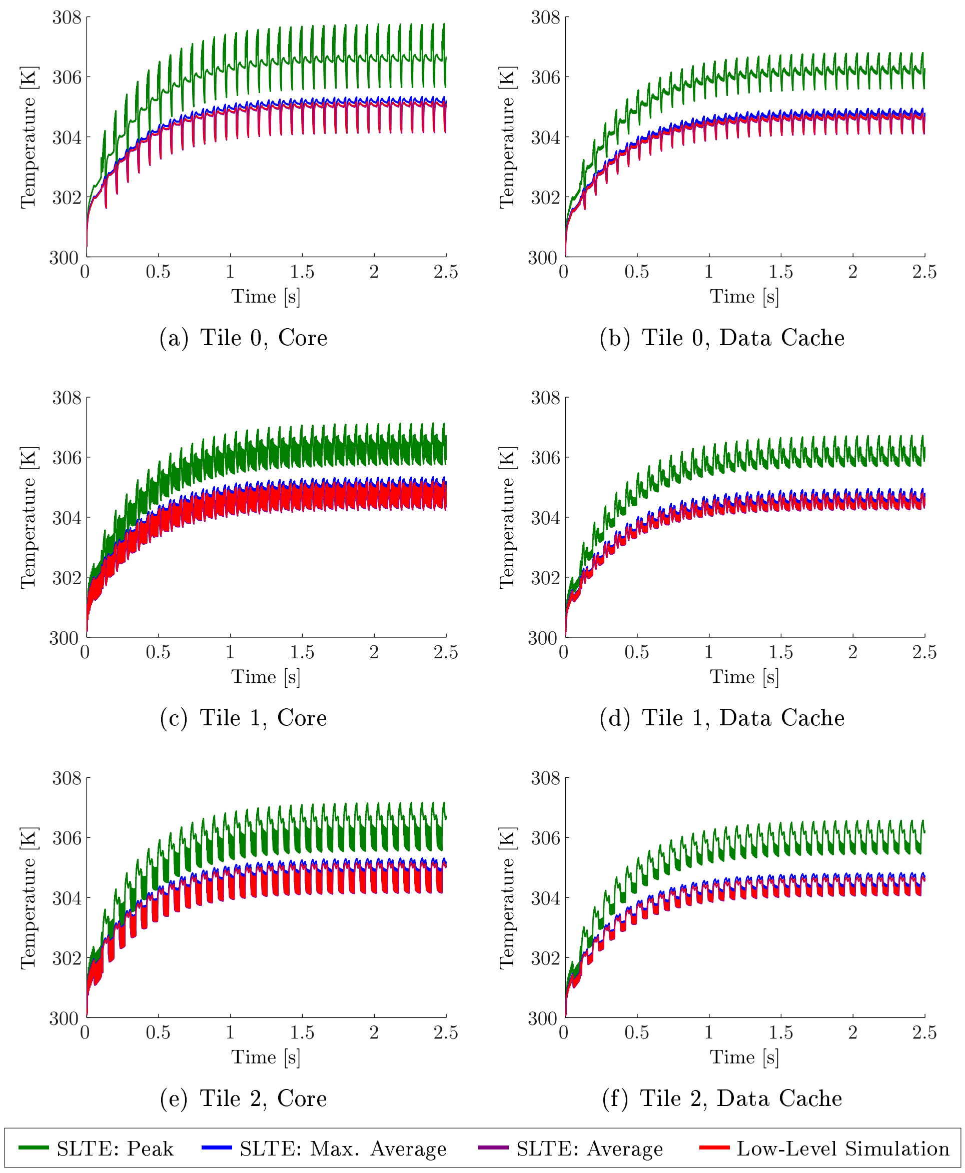


Figure 7.3: Matrix Multiplication, Mapping I:
Temperature evolution of three tiles of the MPARM virtual platform when the matrix multiplication application is executed. For each tile, both the core and data cache is outlined. The application has been simulated with SLTE using three different power values, that is, the average, maximum average, and peak power consumption. As the temperature evolution of SLTE with the average power consumption as power value is almost indistinguishable from the temperature evolution recorded from the low-level simulation, their lines fall on each other.

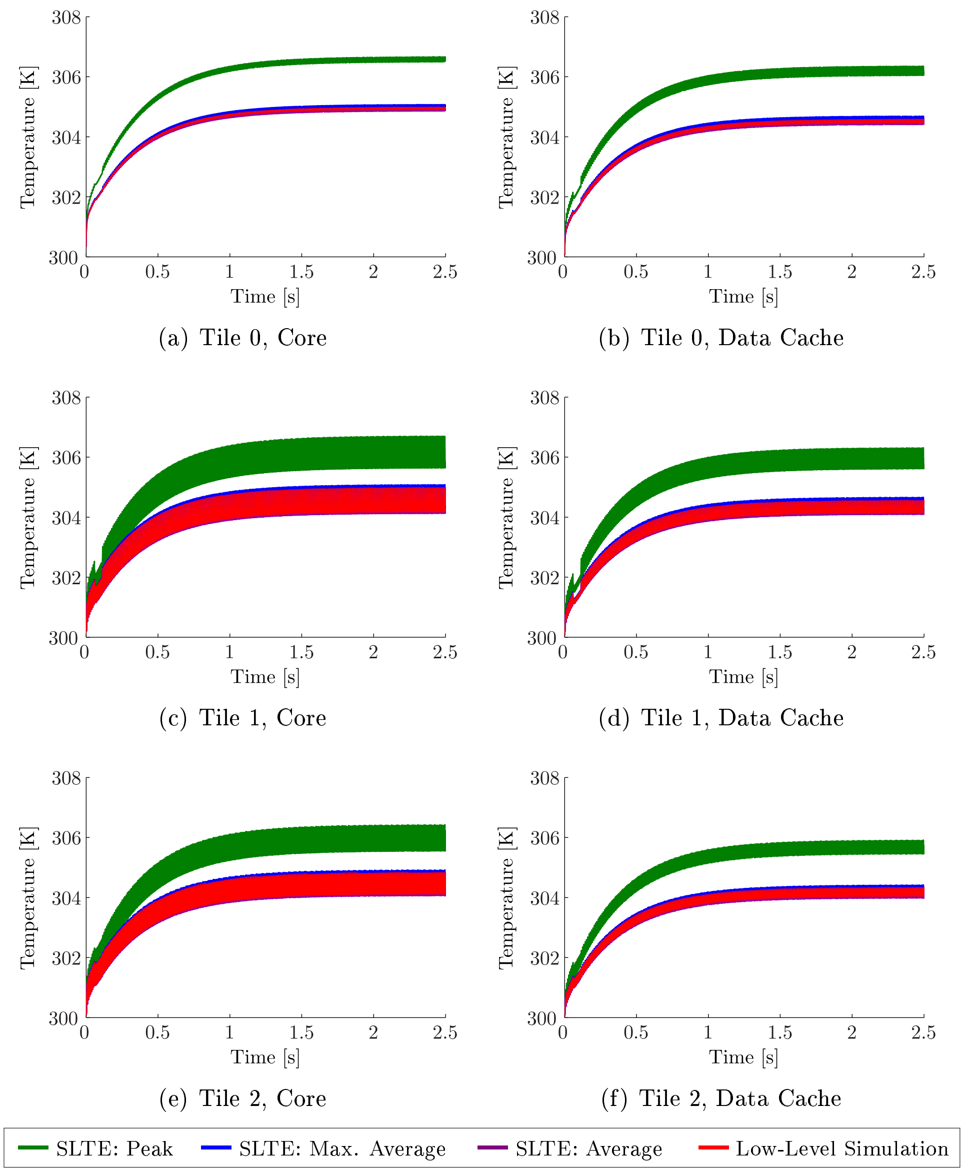


Figure 7.4: Matrix Multiplication, Mapping VII:
Temperature evolution of three tiles of the MPARM virtual platform when the matrix multiplication application is executed. For each tile, both the core and data cache is outlined. The application has been simulated with SLTE using three different power values, that is, the average, maximum average, and peak power consumption. As the temperature evolution of SLTE with the average power consumption as power value is almost indistinguishable from the temperature evolution recorded from the low-level simulation, their lines fall on each other.

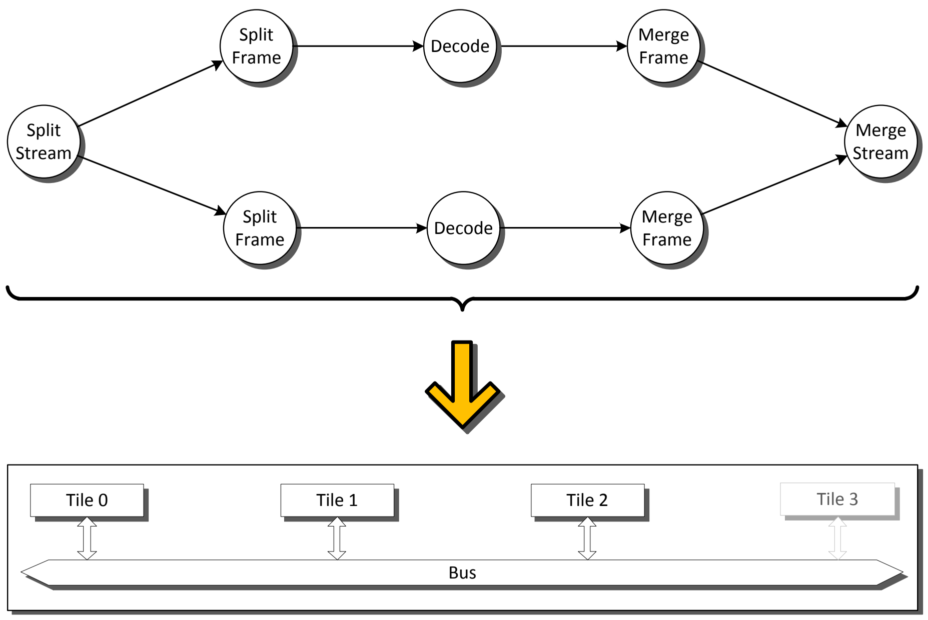


Figure 7.5: Process network of a distributed implementation of the MJPEG decoder when two frames are decoded in parallel.

7.2.3 Motion JPEG Decoder

The third application that is considered in the evaluation of SLTE is an MJPEG decoder [80]. MJPEG is a video codec in which each video frame is separately compressed as a JPEG image. The process network of the decoder considered in this evaluation is outlined in Fig. 7.5. The decoder's input is a video stream that is first split into its individual frames. The second subactor splits the frames into macroblocks, which are separately decoded. The decoded macroblocks are stitched together into a frame and finally into a stream. To improve the performance of the application, various frames can be decoded in parallel as outlined for the case of two frames in Fig. 7.5.

We consider two different parallelizations in this evaluation. In the first scenario, no parallelization is considered, that is, only one frame is decoded per time. As the process network only consists of totally five processes, we map the whole application onto a distributed platform with three tiles and its characteristics are summarized in Table 7.7. In the second scenario, two frames are decoded in parallel and the application is mapped onto a distributed architecture of four tiles. Again, pessimism, the execution times, and the speed-up are reported in Table 7.8. In both scenarios, a stream of eight frames is decoded during one run of the system.

Table 7.7: MJPEG Decoder, One Frame, Different Mappings:
Pessimism, execution times and speed-ups of the first evaluation scenario where four different mappings are compared. The application is mapped onto a distributed architecture of three tiles and the mapping that is used to perform the model calibration is marked with a star.

	Pessimism			**Execution Time**		
	Average	*Min*	*Max*	*Platform*	*SLTE*	*Speed-up*
Mapping 1*	0.0014	0.0002	0.0074	1.46 s	6.38 s	394
Mapping 2	0.0054	0.0004	0.0182	1.46 s	5.99 s	408
Mapping 3	0.0021	0.0006	0.0081	1.49 s	6.24 s	406
Mapping 4	0.0019	0.0002	0.0114	1.48 s	5.95 s	409

Table 7.8: MJPEG Decoder, Two Frames In Parallel, Different Mappings:
Pessimism, execution times and speed-ups of the second evaluation scenario where four different mappings are compared. The application is mapped onto a distributed architecture of four tiles and the mapping that is used to perform the model calibration is marked with a star.

	Pessimism			**Execution Time**		
	Average	*Min*	*Max*	*Platform*	*SLTE*	*Speed-up*
Mapping 1*	0.0023	0.0000	0.0128	1.45 s	6.99 s	410
Mapping 2	0.0028	0.0001	0.0148	1.45 s	6.99 s	411
Mapping 3	0.0028	0.0011	0.0050	1.44 s	7.64 s	373
Mapping 4	0.0012	0.0001	0.0081	1.40 s	7.94 s	391

To demonstrate the viability of SLTE in estimating the transient temperature evolution of an application, the temperature evolution of totally twelve components are plotted when the MJPEG decoder is executed. Figures 7.6 and 7.7 plot the temperature evolution of the first and second tile, and the third and forth tile, respectively. The system corresponds to the first mapping of Table 7.8. Again, the temperature evolution computed by SLTE is plotted when the average, maximum average, and the peak power consumptions are used to calibrate the model. The temperature of the low-level simulation is plotted as reference.

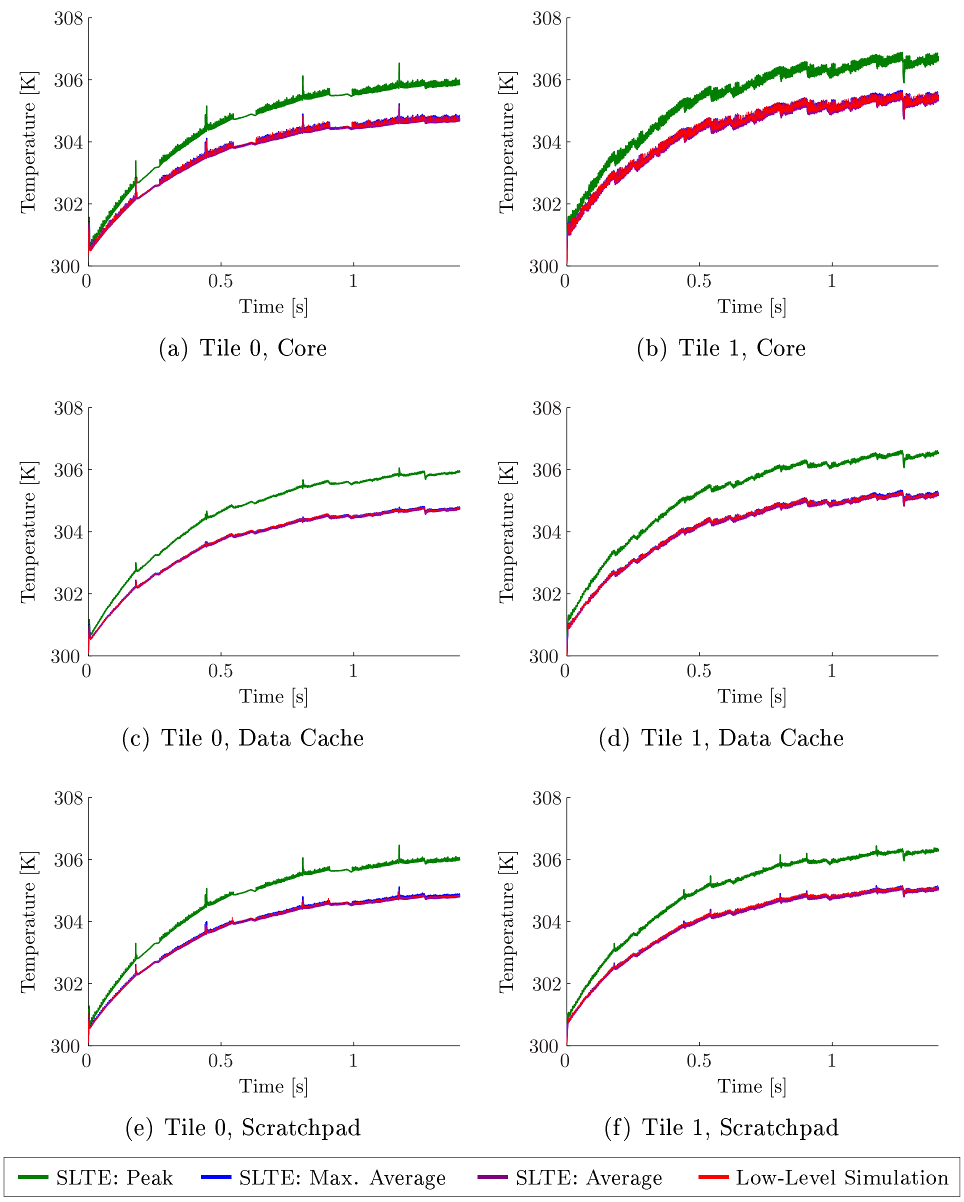
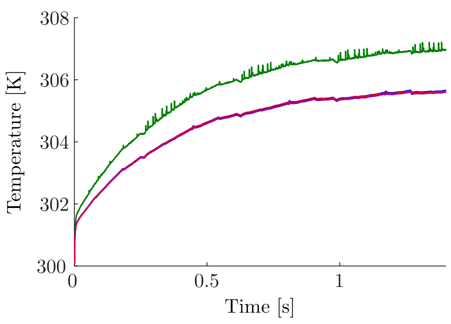
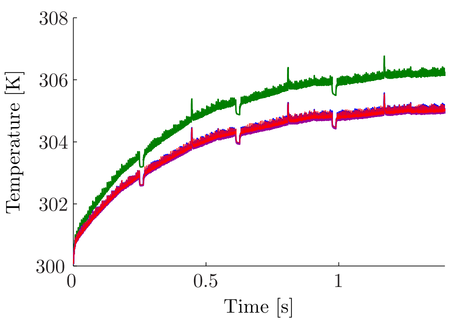


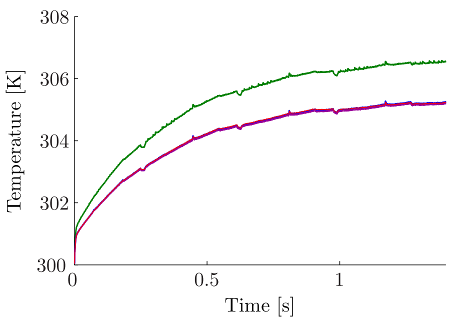
Figure 7.6: MJPEG Decoder, Mapping I, Tile 1 & 2:
Temperature evolution of two tiles of the MPARM virtual platform when the MJPEG decoder application is executed. Three components, namely the core, data cache and scratchpad, are outlined for each tile. The application has been simulated with SLTE using three different power values, that is, the average, maximum average, and peak power consumption.



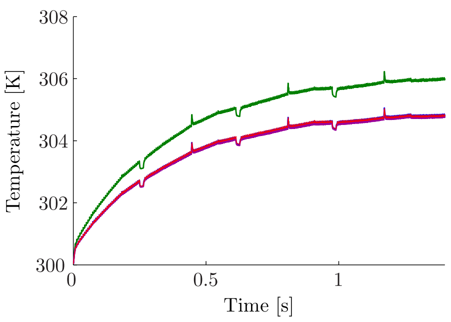
(a) Tile 2, Core



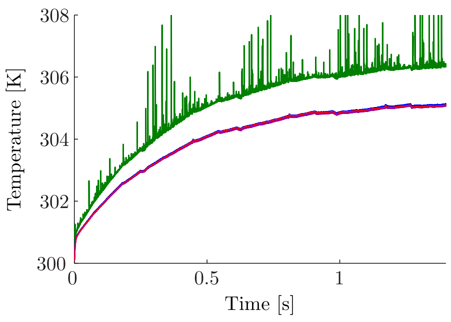
(b) Tile 3, Core



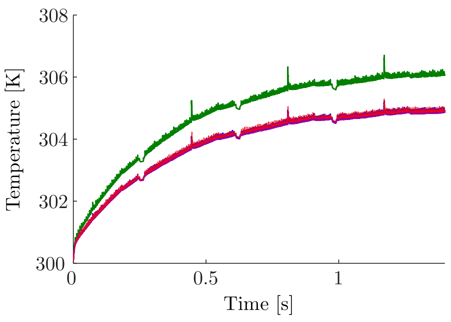
(c) Tile 2, Data Cache



(d) Tile 3, Data Cache



(e) Tile 2, Scratchpad



(f) Tile 3, Scratchpad

🟩 SLTE: Peak 🟦 SLTE: Max. Average 🟪 SLTE: Average 🟥 Low-Level Simulation

Figure 7.7: MJPEG Decoder, Mapping I, Tile 3 & 4:
Temperature evolution of two tiles of the MPARM virtual platform when the MJPEG decoder application is executed. Three components, namely the core, data cache and scratchpad, are outlined for each tile. The application has been simulated with SLTE using three different power values, that is, the average, maximum average, and peak power consumption.

7.3 Discussion

To evaluate a multi-processor streaming application already in an early design stage, automatically calibrated thermal evaluation models are required whose input are the software synthesis specifications. Having such a model, we aimed to achieve an evaluation time that is faster as the system's execution time on the real platform. Therefore, many mappings can be evaluated during design space exploration without having too much computational overhead. Furthermore, an adequate accuracy is required, that is, the pessimism should be less than one percent.

7.3.1 Accuracy

In all results listed in Tables 7.2 to 7.8, the average pessimism is less than one percent. From the group of experiments, that use the same mapping for model calibration and analysis, the pessimism is even less than 0.36 %. Only five experiments have a maximum observed pessimism that is higher than one percent, however, the pessimism never exceeds 1.5 %. To interpret these results, we first transform them to absolute temperature values. In our experiments, roughly 0.2 K corresponds to a pessimism of one percent. Therefore, the absolute error of the peak temperature does never exceed 0.3 K. The inaccurate estimation of the temperature of individual components can be attributed to the following two factors:

1. While the power consumption is approximately constant for computing components, the power curve of other components swings much more during a segment. Consider, for example, the local memory, an application only consumes power if the required data is not yet cached. Therefore, their power consumption is unequally distributed over time, which results in a higher peak temperature.

2. In all experiments, only one benchmark mapping has been used to calibrate the system. However, the accuracy strongly depends on the selection of the benchmark mapping. Every process or even segment uses a different characteristic amount of time to restore and store its context and only these characteristics that are actually observed during model calibration can be accurately modeled in SLTE.

The comparison of the results of the producer-consumer and matrix multiplication application makes trends even more clear. On the one hand, execution times that are small often lead to many context switches and therefore, higher inaccuracy. On the other hand, the constant power consumption approximation leads to an inaccuracy in estimating the temperature of applications with segments that have a long execution time.

7.3.2 Speed-up

In both the producer-consumer application and the matrix multiplication application, the time to evaluate a system in SLTE is always lower than the execution time of the system on the real platform. This means that the thermal characteristics of the system are obtained in real time. However, evaluating the MJPEG decoder in SLTE takes four to five times the execution time of the system on the real platform. The average speed-up of all 41 experiments is 1939. It is noticeable, that the speed-up of the MJPEG decoder is very low compared to all other applications. To interpret this result, we need to consider the code structure of the MJPEG decoder. The code is assembled of a large number of small segments that only forward some data from one node to another node. Furthermore, as only eight frames have been decoded in all experiments, no repetitive behavior of the decoder can be observed that would reduce the time to evaluate the application in SLTE.

The experiments can be categorized in two groups, with either the computation of the power consumption or the computation of the temperature as bottle neck. All experiments involving the MJPEG decoder are examples of the first group. The time to calculate the power consumption mainly depends on the actual execution time of the segments. Therefore, an approach to improve the time to calculate the power consumption would be to unite short segments that logically belong together into one large segment. In Listing 7.1, the reading of a large array is split into various small segments. In the current version of SLTE, this code would generate two times `NB_SEND` segments. As the influence of the *computing* segments on the overall power consumption is negligible, a viable approach is to unite all these short segments into a large segment. However, its impact on the accuracy of the temperature is an open question.

Listing 7.1: Example code to illustrate the idea of unite short segments into a large segment to improve the computing time of the power consumption in SLTE.

```
for (int i = 0; i < NB_SEND; i++) {
    read(( void *) PORT_IN, &buffer[MAX_SEND * i]);
}
```

The evaluation of various mappings of the matrix application is an example of the second group. The speed-up is mainly bounded by the time to compute the temperature out of the power consumption. The easiest approach to reduce this time is the selection of a higher sampling interval, however, this often results in reduced accuracy. Another approach to reduce the time to evaluate a system is the use of model reduction [39, 81]. Nonetheless, both approaches do not require a conceptual change of our model.

7.4 Summary

This chapter described various case studies to evaluate the proposed thermal emulation techniques for estimating the transient temperature evolution of a multi-processor streaming application. In all three case studies, we considered streaming applications that are described as process networks and mapped onto a distributed platform. It was shown that SLTE has remarkable advantages over traditional low-level simulation methods in the design space exploration.

A prototype implementation of SLTE in DOL is used to demonstrate the viability of the proposed approaches. It was shown that speed-ups of more than 2000 are achieved over a low-level thermal evaluation tool chain by an overall reduction of the accuracy of less than one percent. Finally, the MJPEG decoder application was used to highlight the fact that the speedup is reduced if the application consists of segments that are much smaller than the sampling period.

8 Thermal Simulation: Conclusion and Outlook

8.1 Conclusion

Multi-processor streaming applications with real-time requirements demand system-level methodologies that guarantee both thermal constraints and real-time deadlines at the design time. The first part of this master thesis tackles this challenge and proposes techniques to estimate the transient temperature behavior of a many-core system in an early design stage.

Compositional thermal evaluation methods outperform traditional, low-level based evaluation methods due to a much faster execution time when comparing thousands of design alternatives. However, compositional thermal evaluation models require timing and thermal parameters to model the behavior of the application. The first contribution of this part of the thesis is the design and implementation of a thermal evaluation tool chain to automatically calibrate compositional thermal evaluation models. The tool chain is made up of three fundamental tools, namely a synthesis tool, a low-level virtual platform, and a thermal simulator. The viability of the tool chain is assessed by a prototype implementation based on DOL, the MPARM virtual platform, and HotSpot. The MPARM virtual platform has been extended to support the computation of the transient power dissipation, and to differ between static and dynamic power consumptions in order to tackle the challenge of temperature-dependent static power dissipation.

The second contribution is the design of SLTE, a compositional thermal evaluation model. It emulates the transient temperature evolution of a distributed system that is described by its high-level specification, namely the application, architecture, and mapping specification, and additional thermal and timing parameters. The calculation of the transient temperature evolution is made up of three parts. First, the scheduling of a system is calculated with a trace-based simulator. Afterwards, the scheduling is annotated with the corresponding power consumption and finally, the temperature is calculated with a method to partly precompute the solution of the differential equation system.

A prototype implementation of SLTE in DOL is used to evaluate the model in three case studies. It shows that SLTE provides speed-ups in the order of three magnitudes over the low-level tool chain while having an average error of less than one percent. So far, hardware/software emulation frameworks that are based on a Field Programmable Gate Array (FPGA) emulation platform have been the state-of-the-art method for thermal analysis in terms of evaluation time. However, SLTE achieves similar speed-ups and accuracies, but is much more flexible and cheaper.

While providing detailed information about the transient temperature characteristics of a multi-processor streaming application, thermal simulation has the drawback that it only covers a fraction of all possible system behaviors. Therefore, it cannot be used to determine hard bounds on the temperature as it is often required by modern embedded real-time systems. In the second part of this thesis, we will tackle this drawback and propose an analytic method to calculate the worst-case peak temperature of a many-core system.

8.2 Outlook

As thermal simulation methods are indispensable in an early design stage of a multi-processor streaming application, future research should cover all the discussed aspects and there are still several interesting topics for supplementary research.

First of all, the proposed compositional thermal analysis method should be extended to support other platforms and architectures. Basically, this includes two steps. On the one hand, the automated model calibration method has to be transferred to the new platform to support the extraction of the required timing and thermal parameters. On the other hand, the actual emulation model has to be extended with an accurate model of every novel type of component like Network-on-a-Chip (NoC) or Dynamic Random Access Memory (DRAM). In particular, the transformation of the proposed models to a real platform would be interesting to investigate, as the approximated temperature could be compared with the actual temperature of the hardware. Examples of such platforms include the newly announced Intel Single-chip Cloud Computer (SCC) [2] platform or the Platform 2012 [6] that even supports three-dimensional stacking.

Part II

Analytic Thermal Analysis

9 Analytic Thermal Analysis: An Introduction

9.1 Overview

In hard real-time systems, the completion of a task after its deadline is considered as useless as it may cause a failures of the system. Therefore, such systems require safe bounds on timing properties, and, as the predictability of a system highly depends on its temperature, safe bounds on the temperature. Consequently, the identification of the worst-case peak temperature is a crucial task in the design of a multi-processor streaming application for modern embedded real-time systems.

Thermal simulation methods, as considered in the first part of this master thesis, only cover a fraction of all possible system behaviors, and therefore, they cannot be used to determine hard bounds on the temperature. In the second part of this thesis, we discuss the question of obtaining the worst-case peak temperature of a multi-processor streaming application that is mapped onto a distributed architecture by using analytic worst-case analysis methods.

Consequently, we propose and evaluate a method to calculate the worst-case peak temperature of a many-core system. Based on a former work to calculate the worst-case peak temperature of a single-node system, we derive in Chapter 10 an analytic worse-case peak temperature method for many-core systems. A concrete implementation of the method is presented for event streams described by the period-jitter-delay model. Afterwards, in Chapter 11, the method is applied to various case studies and two approximation algorithms that tremendously reduce the evaluation time, are discussed and evaluated.

9.2 Related Work

To the best of our knowledge, this work is the first framework that calculates a tight upper bound on the worst-case peak temperature of a multi-processor streaming application. Current approaches to analyze the temperature characteristics of a multi-processor application mainly include solutions based on simulation and exclude analytic methods. A broad overview of thermal simulation methods is given in the first part of this master thesis, in Section 2.2. In addition to this summary, the first part of this thesis presented methods to simulate the temporal temperature evolution of a multi-processor streaming application.

Providing hard bounds of the analysis parameters is nowadays essential for real-time embedded systems. In the last decade, various techniques for analytical performance analysis have been proposed. Compositional best-case/worst-case methods like Modular Performance Analysis (MPA) [5] or Symbolic Timing Analysis for Systems (SymTA/S) [61] provide upper and lower bounds of the timing parameters of a distributed application. Recently, Rai et al. [15] proposed a method to extract the worst-case peak temperature of a single-core system. Its extension to multi-core systems will be discussed in the following.

10 Worst-Case Peak Temperature of a Many-Core System

The derivation of a framework to obtain the worst-case peak temperature of a multi-processor streaming application that is mapped onto a distributed architecture, is the topic of this chapter. The work is an extension of [15], which calculates the worst-case peak temperature of a single-node system. However, as the thermal influence of one node on another node is delayed, new approaches are required to tackle the problem. The proposed framework considers an event model based on arrival curves of network and real-time calculus. Its only requirement towards the processing components is that they are work-conserving, that is, they have to process an event if there is at least one event in the ready queue. To the best of our knowledge, this work is the first framework that calculates a tight upper bound on the worst-case peak temperature of a multi-processor streaming application.

After presenting the basic thermal and computational model in Section 10.1, Section 10.2 introduces the thermal analysis model and provides an optimization problem that constructs the worst-case accumulated computing time function. In Section 10.3, the optimization problem is solved for event streams described by the period-jitter-delay model. The chapter concludes with a short summary.

10.1 System Model

Based on the previous explanation of a thermal model and the ideas of arrival curves [82] for modeling the event streams, we introduce the basic models to calculate the worst-case peak temperature of a multi-processor system in this section. However, to establish a common language, we first introduce a few basic terms of the linear algebra.

Definition 1. *A real $n \times n$ matrix* $\mathbf{G}$ *is called* non-negative *if and only if*

$$g_{i,j} \geq 0 \tag{10.1}$$

for all i, j.

Definition 2. *A real $n \times n$ matrix* $\mathbf{G}$ *is called* non-positive *if and only if*

$$g_{i,j} \leq 0 \tag{10.2}$$

for all i, j.

Definition 3. *A real $n \times n$ matrix* $\mathbf{G}$ *is called* essentially non-negative *if and only if*

$$g_{i,j} \geq 0 \qquad i \neq j. \tag{10.3}$$

10.1.1 Thermal Model

The duality between the heat transfer and an electrical network has been used to derive a thermal model in Sections 3.3.2 and 4.3. Similarly, in this chapter, we model the transient temperature behavior of a multi-processor system as a multi-dimensional first-order differential equations:

$$\mathbf{C} \cdot \frac{\mathrm{d}\mathbf{T}}{\mathrm{d}t} = (\mathbf{P} + \mathbf{S} \cdot \mathbf{T}_{amb}) - (\mathbf{G} + \mathbf{S}) \cdot \mathbf{T} \tag{10.4}$$

where $\mathbf{C}$ is the thermal capacitance matrix, $\mathbf{G}$ the thermal conductance matrix, $\mathbf{S}$ the thermal ground conductance matrix, $\mathbf{P}$ the power dissipation vector, $\mathbf{T}$ the temperature vector, and $\mathbf{T}_{amb} = T_{amb} \cdot [1, \ldots, 1]'$ the ambient temperature vector. $\mathbf{G}$ is a non-positive matrix whose diagonal elements are 0 and $\mathbf{S}$ is a non-negative diagonal matrix. Only the power $\mathbf{P}$ and the temperature $\mathbf{T}$ are assumed to be time-variant, however, we will not present the time variable for $\mathbf{P}$ and $\mathbf{T}$ if the context is clear.

From Section 3.3.3, we know that the power depends exponentially on the temperature and that a first-order linear approximation can be used to obtain a fairly accurate model of the temperature-dependency in the normal operating temperature range of modern ICs [56]. Therefore, in this chapter, we describe the temperature dependency of the static power by means of a linear approximation as

$$\mathbf{P}(\mathbf{T}) = \boldsymbol{\phi} \cdot \mathbf{T} + \boldsymbol{\psi} \tag{10.5}$$

where $\boldsymbol{\phi}$ is a diagonal matrix with constant coefficients, and $\boldsymbol{\psi}$ a vector with constant coefficients. Therefore, we can express the jth component of $\mathbf{P}$ as $P_j(T) = \phi_{jj} \cdot T_j + \psi_j$. We only distinguish between an 'active' and an 'idle' mode of the processing components and denote the power of component j in the 'active' mode as P_j^a and the corresponding power in the 'idle' mode as P_j^i. Due to the fact that only the coefficient ψ depends on the mode of the processing component, we characterize the power of the processing modes as

$$P_j^a(T) = \phi_{jj} \cdot T_j + \psi_j^a \tag{10.6}$$

and

$$P_j^i(T) = \phi_{jj} \cdot T_j + \psi_j^i. \tag{10.7}$$

Lemma 4. *Given a thermal model according to (10.4) and (10.5). The model defines a linear and time-invariant system and can be described by its state-space representation*

$$\frac{\mathrm{d}\mathbf{T}}{\mathrm{d}t} = \mathbf{A} \cdot \mathbf{T} + \mathbf{B} \cdot \mathbf{u} \tag{10.8}$$

where $\mathbf{A}$ *and* $\mathbf{B}$ *are constant matrices and* $\mathbf{u} = [u_1, u_2, \ldots, u_m]'$ *is a time-dependent input vector.*

Proof. The state-space representation of a linear system [83] is defined as

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{A}(t) \cdot \mathbf{x} + \mathbf{B}(t) \cdot \mathbf{u} \tag{10.9}$$

where $\mathbf{x} = [x_1, x_2, \ldots, x_k]'$ is the state vector and $\mathbf{u} = [u_1, u_2, \ldots, u_m]'$ the input vector. Furthermore, the system is said to be time-invariant, if both the matrices A and B are constant, that is, they are independent of the time t [83].

Rewriting (10.4) with (10.5) yields

$$\frac{\mathrm{d}\mathbf{T}}{\mathrm{d}t} = -\mathbf{C}^{-1} \cdot (\mathbf{G} + \mathbf{S} - \boldsymbol{\phi}) \cdot \mathbf{T} + \mathbf{C}^{-1} \cdot (\boldsymbol{\psi} + \mathbf{S} \cdot \mathbf{T}_{amb}) \tag{10.10}$$

and by equating the coefficients, we find

$$\begin{aligned} \mathbf{A} &= -\mathbf{C}^{-1} \cdot (\mathbf{G} + \mathbf{S} - \boldsymbol{\phi}) \\ \mathbf{B} &= \mathbf{C}^{-1} \\ \mathbf{x} &= \mathbf{T} \\ \mathbf{u} &= \boldsymbol{\psi} + \mathbf{S} \cdot \mathbf{T}_{amb}. \end{aligned} \tag{10.11}$$

Obviously, both the matrices $\mathbf{A}$ and $\mathbf{B}$ only consist of matrices with constant coefficients, which concludes our proof. □

The following corollaries describe several properties of the matrices used in the state-space representation of the thermal model.

Corollary 5. *Suppose that a thermal model is given according to (10.8). Then the matrix* $\mathbf{B}$ *defined by (10.11) is non-negative.*

Corollary 6. *Suppose that a thermal model is given according to (10.8). Then the matrix*

$$\mathbf{A} = -\mathbf{C}^{-1} \cdot (\mathbf{G} + \mathbf{S} - \boldsymbol{\phi}) \tag{10.12}$$

defined by (10.11) is essentially non-negative.

Proof. First note that $\mathbf{S}$ and $\boldsymbol{\phi}$ are diagonal matrices and $\mathbf{G}$ is a non-positive matrix whose diagonal elements are 0. Therefore, $-(\mathbf{G} + \mathbf{S} - \boldsymbol{\phi})$ is an essentially non-negative matrix. The proof concludes from the facts that $\mathbf{C}$ is a diagonal matrix with positive elements, and the coefficients of an essentially non-negative matrix do not change their algebraic sign after the multiplication with a positive diagonal matrix. □

Note that, for peak temperature analysis, we implicitly assume that the thermal model is BIBO-stable[1].

10.1.2 Computational Model

Targeting distributed multi-processor systems, we assume that the computational model of every processor is independent of each other, but all processors stick to the same rules. In the following, we derive the computational model of our multi-processor system from the computational model of single processors. Furthermore, in this thesis, we consider general event arrivals modeled by arrival curves from network and real-time calculus [82, 84].

The computational model of a single processor follows the model described in [15]. For completeness, we will summarize its core statements in the following. While queuing all uncompleted events, a component is assumed to processes at most $t - s$ time units in a time interval $[s, t)$. The only restriction towards the processing component is work-conserving, that is, if there is at least one event in the ready queue, the processing component has to process it. We call a system work-conserving if all its components are work-conserving.

[1]BIBO-stability requires that every impulse response of the system is absolute integrable, and therefore

$$\int_{-\infty}^{\infty} |h(t)| \, \mathrm{d}\, t < \infty. \tag{10.13}$$

In the time interval $[s, t)$, events with a total workload of $R(s, t)$ time units arrive. The arrival curve α upper bounds the cumulative workload, where

$$R(s,t) \leq \alpha(t-s) \quad \forall s < t \tag{10.14}$$

with $\alpha(0) = 0$. Note that the arrival curve is sub-additive [82], if it satisfies

$$\alpha(x) + \alpha(y) \geq \alpha(x+y), \quad \forall x, y \geq 0. \tag{10.15}$$

Suppose that several independent workload functions $R^{(k)}$ are bounded individually by arrival curves $\alpha^{(k)}$. For the case that the workload functions are concurrently processed in a single component, the accumulated workload can be bounded as follows [15]:

$$R(s,t) \leq \sum_{\forall k} \alpha^{(k)}(t-s) = \alpha(t-s). \tag{10.16}$$

The accumulated computing time Q describes the time a component is spending to compute the incoming workload of $R(s, t)$ time units. Therefore, for work-conserving scheduling algorithms, the accumulated computing time $Q(s, t)$ of the component in the time interval $[s, t)$ can be expressed by

$$Q(s,t) = \inf_{s \leq u \leq t} \{(t-u) + R(s,u)\} \tag{10.17}$$

provided that there is no buffered workload in the ready queue at time s. Using the upper bound of the cumulative workload R as defined in (10.14), the accumulated computing time $Q(t-\Delta, t)$ can be upper bounded by $\gamma(\Delta)$ for intervals with length Δ:

$$Q(t-\Delta,t) \leq \gamma(\Delta) = \inf_{0 \leq \lambda \leq \Delta} \{(\Delta - \lambda) + \alpha(\lambda)\}. \tag{10.18}$$

For any fixed s with $s < t$, the accumulated computing time $Q(s, t)$ is monotonically increasing and has either slope 1 or 0. The mode function $S(t)$ determines the operating mode of the processing component and can be expressed by

$$S(t) = \frac{\mathrm{d}Q(s,t)}{\mathrm{d}t} \in \{0, 1\}. \tag{10.19}$$

Therefore, the processing component is in 'active' mode if $S(t) = 1$ and in 'idle' mode if $S(t) = 0$.

Suppose that we totally have n components, and every component j receives in a time interval $[s, t)$ a cumulative workload trace of $R_j(s, t)$, the cumulative workload trace of a multi-processor system is defined by

$$\mathbf{R}(s,t) = \begin{pmatrix} R_1(s,t) \\ \vdots \\ R_n(s,t) \end{pmatrix} \tag{10.20}$$

with $R_j(s,t) \leq \alpha_j(t-s)$ for all $s < t$ and $1 \leq j \leq n$. Similarly, we define the accumulated computing time function $\mathbf{Q}$ of a multi-processor system by

$$\mathbf{Q}(t-\Delta, t) = \begin{pmatrix} Q_1(t-\Delta,t) \\ \vdots \\ Q_n(t-\Delta,t) \end{pmatrix} \tag{10.21}$$

where Q_j is the accumulated computing time function of component j and can be expressed by

$$Q_j(t-\Delta,t) \leq \gamma_j(\Delta) = \inf_{0 \leq \lambda \leq \Delta} \{(\Delta - \lambda) + \alpha_j(\lambda)\}\,. \tag{10.22}$$

In the following, we implicitly assume that the schedulability test for real-time systems has been passed, the curve α is sub-additive, and no thermal management is applied to reduce the temperature.

10.1.3 Problem Statement

The remainder of this chapter addresses the question of determining the worst-case peak temperature of a multi-processor streaming application. Therefore, the problem can be formulated as follows:

> *Given a work-conserving system that is characterized by a stable thermal model, the aim is to determine the worst-case peak temperature* $\mathbf{T}^*$ *for any cumulative workload* $\mathbf{R}$ *as defined in (10.20).*

10.2 Thermal Analysis

In order to calculate an upper bound of the peak temperature T_j^* of component j, we first reduce the problem to determine an upper bound of the temperature if all except one power source are eliminated. Consequently, we provide an optimization problem, that constructs the accumulated computing time function $Q^*(0,\Delta)$ that leads to the maximum temperature T_j^*. Finally, we summarize various statements, which result from the general optimization problem.

10.2.1 Temperature Superposition

The output of a linear system is the sum of two independent system responses, namely the zero-input response and the zero-state response. The zero-input response of a linear system characterizes the initial conditions, and the zero-state response is the output of the system if the initial conditions are neglected. Suppose that the system starts at time t_0, the temperature vector at time t can be expressed as

$$\mathbf{T}(t) = \mathbf{T}_{\text{zero-input}}(t) + \mathbf{T}_{\text{zero-state}}(t). \tag{10.23}$$

Therefore, the worst-case computing time, that is, the computing time that leads to the highest temperature at a time τ, is independent of the initial temperature. Consequently, in a first step, we assume that the initial conditions are neglected and the system starts to work at time $t_0 = 0$. However, we will not present the indices and implicitly assume that $\mathbf{T}$ denotes the zero-state response.

Suppose that the considered system is a multi-processor with totally n processing components. Without loss of generality, in the following, we determine the worst-case peak temperature of component i. As a first prerequisite, we reduce the problem of determining the worst-case peak temperature of component i to n simpler problems where we have zero input at all except one components.

Lemma 7. *Given a stable thermal model according to (10.8). Suppose that the computational model of every processor is independent of each other and $T^*_{i,j}(\tau)$ is the worst-case peak temperature of component i at time τ if component j is the only component that has an input unequal zero, that is, $\mathbf{u} = [0, \ldots, 0, u_j, 0, \ldots, 0]'$. Then, the worst-case peak temperature of component i at time τ is*

$$T^*_i(\tau) = \sum_{\forall j} T^*_{i,j}(\tau). \tag{10.24}$$

Proof. The superposition principle states that the output response of a system to a sum of inputs is the sum of the responses to the individual inputs. As it is valid for every linear system [85], we can divide the input $\mathbf{u}$ of our system as follows:

$$\mathbf{u} = \boldsymbol{\psi} + \mathbf{S} \cdot \mathbf{T}_{amb} = \mathbf{u}^{(1)} + \ldots + \mathbf{u}^{(n)} \tag{10.25}$$

where

$$\mathbf{u}^{(k)} = [0, \ldots, 0, \psi_k + s_{kk} \cdot T_{amb}, 0, \ldots, 0]'. \tag{10.26}$$

From the superposition principle follows that the temperature $\mathbf{T}(t)$ at time t can be written as

$$\mathbf{T}(t) = \mathbf{T}^{\mathbf{u}^{(1)}}(t) + \mathbf{T}^{\mathbf{u}^{(2)}}(t) + \ldots + \mathbf{T}^{\mathbf{u}^{(n)}}(t) \tag{10.27}$$

with $\mathbf{T}^{\mathbf{u^{(k)}}}(t)$ the response of the system to input $\mathbf{u}^{(k)}$. Then, the worst-case peak temperature at time τ of component i can be expressed by

$$\begin{aligned} T_i^*(\tau) &= \max_{\mathbf{u}\in\mathbf{U}} (T_i(\tau)) \\ &= \max_{\mathbf{u}\in\mathbf{U}} \left(T_i^{\mathbf{u}^{(1)}}(\tau) + T_i^{\mathbf{u}^{(2)}}(\tau) + \ldots + T_i^{\mathbf{u}^{(n)}}(\tau)\right) \\ &= \max_{\mathbf{u}^{(1)}\in\mathbf{U}^{(1)}} \left(T_i^{\mathbf{u}^{(1)}}(\tau)\right) + \max_{\mathbf{u}^{(2)}\in\mathbf{U}^{(2)}} \left(T_i^{\mathbf{u}^{(2)}}(\tau)\right) + \ldots \\ &\quad + \max_{\mathbf{u}^{(n)}\in\mathbf{U}^{(n)}} \left(T_i^{\mathbf{u}^{(n)}}(\tau)\right) \\ &= \sum_{\forall j} T_{i,j}^*(\tau) \end{aligned} \tag{10.28}$$

where $\mathbf{U}$ is the set of all possible inputs $\mathbf{u}$ and $\mathbf{U}^{(k)}$ is the set of all possible inputs of input $\mathbf{u}^{(k)}$. The second step follows from the fact that all inputs $\mathbf{u}^{(k)}$ are independent of each other. □

Remark 8. *In case that the computational model of every processor is not independent of each other, the worst-case peak temperature of component i at time τ is upper bounded by*

$$T_i^*(\tau) \le \sum_{\forall j} T_{i,j}^*(\tau). \tag{10.29}$$

10.2.2 Worst-Case Computing Time

Motivated by Lemma 7, we calculate the worst-case peak temperature $T_{i,j}^*$ of component i for the input $\mathbf{u}^{(j)} = [0, \ldots, 0, \psi_j + s_{jj} \cdot T_{amb}, 0, \ldots, 0]'$. From Lemma 4 we know, that the considered thermal model is a linear time-invariant system. Suppose that $y(t)$ is the output, $x(t)$ the input and $h(t)$ the impulse response[2] of the system. Then the zero-state output of the

[2]Suppose that the thermal model is defined by (10.8), the matrix of all impulse responses is given by

$$\mathbf{h}(t) = \mathcal{L}^{-1}\left[(s\cdot\mathbf{I} - \mathbf{A})^{-1}\cdot\mathbf{B}\right] \tag{10.30}$$

where $\mathcal{L}^{-1}$ denotes the inverse Laplace transformation. Another point of view of the impulse response is to consider the Single-Input and Multiple-Output (SIMO) system, whose state space model is $\dot{\mathbf{x}} = \mathbf{A}\cdot\mathbf{x} + \mathbf{b}\cdot\mathbf{u}$. Then the impulse response is the solution of the following homogeneous differential equation system:

$$\dot{\mathbf{x}} = \mathbf{A}\cdot\mathbf{b}, \quad \mathbf{x}(0) = \mathbf{b}. \tag{10.31}$$

Therefore, the solution of (10.31), that is, the matrix of impulse responses can be expressed by

$$\mathbf{h}(t) = e^{\mathbf{A}\cdot t}\cdot\mathbf{B}. \tag{10.32}$$

system is described by

$$y(t) = x(t) * h(t). \tag{10.33}$$

Therefore, the temperature $T_{i,j}$ of component i is given as

$$T_{i,j}(t) = u_j^{(j)}(t) * h_{ij}(t) \tag{10.34}$$

where $u_j^{(j)}$ is the jth component of $\mathbf{u}^{(j)}$ and $h_{ij}(t)$ is the impulse response of the jth input to the ith output of the system. Note that, for notation simplicity, we will not present the indices in the following, that is, $u \equiv u_j^{(j)}$, $h(t) \equiv h_{ij}(t)$ and $T(t) \equiv T_{i,j}(t)$. In addition, we name the input by u^a if the processing component is in 'active' mode, and by u^i if it is in 'idle' mode. Throughout this chapter, we will implicitly assume that $u^a \geq u^i$.

Theorem 9. *Suppose that $T(t)$ is the temperature at time instant t for an arbitrary feasible workload trace that is bounded by the arrival curve α. Furthermore, the accumulated computing time function $Q^*(0,\Delta)$ for all $0 \leq \Delta \leq \tau$ leads to temperature $T^*(\tau)$ at time τ and its mode function*

$$S^*(\Delta) = \frac{\mathrm{d}Q(0,\Delta)}{\mathrm{d}\Delta} \in \{0,1\} \tag{10.35}$$

maximizes the function $O(S,\tau)$ defined as

$$O(S,\tau) = \int_0^\tau S(\xi) \cdot h(\tau - \xi)\,\mathrm{d}\xi. \tag{10.36}$$

Then $T^*(\tau)$ is an upper bound on the highest temperature at time τ, that is, $T^*(\tau) \geq T(\tau)$.

Proof. As the mode function $S(t)$ is 1 if the system is 'active' and 0 if the system is 'idle', we find

$$u(t) = S(t) \cdot u^a + (1 - S(t)) \cdot u^i. \tag{10.37}$$

Rewriting (10.34), we get:

$$
\begin{aligned}
T(t) &= u(t) * h(t) \\
&= \int_{-\infty}^{\infty} u(\xi) \cdot h(t-\xi)\,\mathrm{d}\xi \\
&= \int_{-\infty}^{\infty} \big(S(\xi) \cdot u^a + (1 - S(\xi)) \cdot u^i\big) \cdot h(t-\xi)\,\mathrm{d}\xi \\
&= \int_{-\infty}^{\infty} \big(u^i + (u^a - u^i) \cdot S(\xi)\big) \cdot h(t-\xi)\,\mathrm{d}\xi \\
&= \int_{-\infty}^{\infty} h(t-\xi) \cdot u^i\,\mathrm{d}\xi + \int_{-\infty}^{\infty} S(\xi) \cdot (u^a - u^i) \cdot h(t-\xi)\,\mathrm{d}\xi \\
&= u^i \cdot \int_{0}^{\infty} h(t-\xi)\,\mathrm{d}\xi + (u^a - u^i) \cdot \int_{0}^{\infty} S(\xi) \cdot h(t-\xi)\,\mathrm{d}\xi \\
&= u^i \cdot \int_{0}^{t} h(t-\xi)\,\mathrm{d}\xi + (u^a - u^i) \cdot \underbrace{\int_{0}^{t} S(\xi) \cdot h(t-\xi)\,\mathrm{d}\xi}_{O(S,t)}\,.
\end{aligned}
\tag{10.38}
$$

In the second last equation we used the assumption that the system is turned on at $t = 0$, and in the last equation the fact that the thermal model represents, by construction, a causal system[3]. Obviously, the maximization of $O(S, t)$ leads to a maximization of the temperature $T(t)$ and we can reach the conclusion. □

In the following, we use various properties of our state-space representation to characterize and approximate the impulse response.

Lemma 10. *Given a stable thermal model according to (10.8). Suppose that* $\mathbf{A}$ *is irreducible and* $h_{ij}(t)$ *is the impulse response of the* j*th input to the* i*th output, then:*

- *The impulse response is always greater than* 0*, that is*

$$h_{ij}(t) \geq 0 \quad \forall t \geq 0. \tag{10.39}$$

- *The impulse response can be approximated as a non-negative function with one local maxima, namely at* $t = 0$ *if* $i = j$*, or at* $t > 0$ *if* $i \neq j$*.*

Proof. The first item directly follows from Lemma 4 in [86] that states that $e^{t \cdot \mathbf{Q}}$ is positive for all $t > 0$ if and only if $\mathbf{Q}$ is essentially non-negative and irreducible. We know from Corollary 6 that $\mathbf{A}$ is non-negative and therefore,

[3]A system is causal if and only if for any pair of input signals $u_1(t)$ and $u_2(t)$ with $u_1(t) = u_2(t)$ for all $t \leq t_0$, the corresponding outputs satisfy $T_1(t) = T_2(t), \forall t \leq t_0$. For a linear time-invariant system, the condition is equivalent to $h(t) = 0, \forall t < 0$.

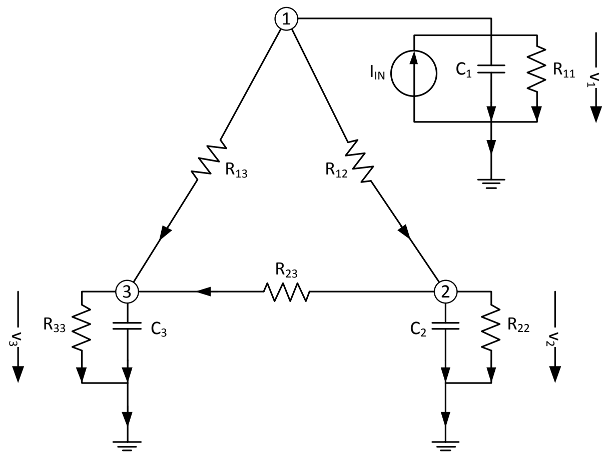


Figure 10.1: RC circuit to support the sketch proof of Lemma 10.

$\boldsymbol{\phi}(t) = e^{t \cdot \mathbf{A}}$ is positive. As the set of impulse responses is the multiplication of $\boldsymbol{\phi}(t)$ with $\mathbf{B}$, which is by Corollary 5 a non-negative matrix, we can reach the conclusion.

Only a sketch proof based on the duality of the heat transfer and an *RC* circuit is provided for the second item. To approximate the impulse response from input j to output i, we remove all current sources in the network except the one that is directly connected to node j. Instead of the impulse response, we first consider the step response and conclude from the properties of the step response to the properties of the impulse response. The step response of a system is the output when its input is a Heaviside step function. The well known duality between the step response $a(t)$ and the impulse response $h(t)$ of a system is given by

$$a(t) = \int_{-\infty}^{t} h(\tau)\,\mathrm{d}\tau \quad \Leftrightarrow \quad h(t) = \frac{\mathrm{d}a(t)}{\mathrm{d}t}. \tag{10.40}$$

Therefore, an inflection point of the step response corresponds to an extremum of the impulse response and vice versa. For the proof, we consider the *RC* circuit outlined in Fig. 10.1, that models a thermal system with three components. The names of the resistances, capacitances, and voltages are denoted in the figure, and the current is named after the component that it flows through. Without loss of generality, we assume that the current source is connected to node 1, and its signal is a Heaviside step function. After an infinite amount of time, the electric potential of every node reaches its steady-state, which is, in a stable system, a finite value. From the thermal perspective, we are only interested in the temporal behavior of the electric potential of the nodes 1, 2, and 3 with respect to the ground, that is, the voltages v_1, v_2 and v_3, respectively.

At first, we note that none of these voltages overshoots its steady-state value as the impulse response is always greater than 0, that is, $h(t) \geq 0$, and consequently, the step response only increases, but never decreases. At time $t = 0^-$, all voltages are 0, and, at time $t = 0^+$, the total amount of current I_{IN} can only flow through the capacitor C_1. Therefore, the voltage v_1 starts to increase according to

$$i_{C_1} = C_1 \cdot \frac{\mathrm{d}v_1(t)}{\mathrm{d}t} \tag{10.41}$$

and approaches steady-state. On the other hand, the voltages v_2 and v_3 start to increase at a slow rate as the resistors R_{12} and R_{13}, respectively, delay their increase. As v_1 approaches steady-state, its increase flattens. Simultaneously, the voltages v_2 and v_3 can increase faster until they flatten towards steady-state. Another inflection point in the transient temperature behavior of the voltages v_1 and v_2 requires that the current $i_{R_{23}}$ would change its direction. However, as both the voltages v_2 and v_3 have to increase monotonically, and approach steady-state, the current $i_{R_{23}}$ would change its direction at the earliest when both voltages have almost reached steady-state. Therefore, we can neglect the additional inflection points of v_2 and v_3, which legitimates our approximations. □

Note that, in the following, we implicitly assume that the impulse response $h(t)$ behaves according to Lemma 10 and denote the time where $h(\tau - t)$ reaches its maximum value with t_m, that is, t_m satisfies the following expressions:

$$\left.\frac{\mathrm{d}h(\tau - t)}{\mathrm{d}t}\right|_{t=t_m} = 0 \quad \text{and} \quad \left.\frac{\mathrm{d}^2h(\tau - t)}{\mathrm{d}t^2}\right|_{t=t_m} < 0. \tag{10.42}$$

Next we will proof that shifting an 'active' processing mode closer to t_m will always increase the temperature at time τ.

Lemma 11. *We consider two mode functions $S(t)$ and $\overline{S}(t)$ defined for $t \in [0, \tau)$. For given $\delta > 0, \sigma \geq 0, \sigma + 2\delta < t_m$, the two mode functions differ as follows:*

- *$S(t) = 1$ for all $t \in [\sigma, \sigma + \delta)$ ('active mode'),*
- *$S(t) = 0$ for all $t \in [\sigma + \delta, \sigma + 2\delta)$ ('idle mode'), and*
- *$\overline{S}(t) = 1 - S(t)$ for all $t \in [\sigma, \sigma + 2\delta)$.*

Then the temperature, denoted as $\overline{T}(\tau)$ at time τ for mode function $\overline{S}(t)$ is not less than the temperature, denoted as $T(\tau)$ at time τ for mode function $S(t)$:

$$\overline{T}(\tau) \geq T(\tau). \tag{10.43}$$

Proof. According to Lemma 10, $h(\tau - t)$ monotonically increases in the interval $[0, t_m)$. By using the result of (10.38), we can express the difference of the temperatures $\overline{T}(\tau)$ and $T(\tau)$ by

$$\begin{aligned}\overline{T}(\tau) - T(\tau) &= \overline{u}(t) * h(t) - u(t) * h(t) \\ &= u^i \cdot \int_0^t h(t-\xi)\,\mathrm{d}\xi + (u^a - u^i) \cdot O(\overline{S}, \tau) \\ &\quad - u^i \cdot \int_0^t h(t-\xi)\,\mathrm{d}\xi + (u^a - u^i) \cdot O(S, \tau) \\ &= (u^a - u^i) \cdot \left(O\left(\overline{S}, \tau\right) - O\left(S, \tau\right)\right)\end{aligned} \tag{10.44}$$

where $\overline{u}(t)$ is the input resulting from mode function $\overline{S}(t)$ and $u(t)$ the input resulting from mode function $S(t)$. As the mode functions satisfies $\overline{S}(t) = S(t)$ for all $t \in [0, \sigma)$ and $t \in [\sigma + 2\delta, \tau)$, we find

$$\begin{aligned}O\left(\overline{S}, \tau\right) - O\left(S, \tau\right) &= \int_0^\tau \overline{S}(\xi) \cdot h(\tau - \xi)\,\mathrm{d}\xi - \int_0^\tau S(\xi) \cdot h(\tau - \xi)\,\mathrm{d}\xi \\ &= \int_\sigma^{\sigma+2\delta} \overline{S}(\xi) \cdot h(\tau - \xi)\,\mathrm{d}\xi - \int_\sigma^{\sigma+2\delta} S(\xi) \cdot h(\tau - \xi)\,\mathrm{d}\xi \\ &= \int_{\sigma+\delta}^{\sigma+2\delta} \overline{S}(\xi) \cdot h(\tau - \xi)\,\mathrm{d}\xi - \int_\sigma^{\sigma+\delta} S(\xi) \cdot h(\tau - \xi)\,\mathrm{d}\xi \\ &= \int_{\sigma+\delta}^{\sigma+2\delta} h(\tau - \xi)\,\mathrm{d}\xi - \int_\sigma^{\sigma+\delta} h(\tau - \xi)\,\mathrm{d}\xi.\end{aligned} \tag{10.45}$$

As $h(\tau - t)$ monotonically increases from 0 to t_m, we finally get

$$O\left(\overline{S}, \tau\right) - O\left(S, \tau\right) \geq 0. \tag{10.46}$$

□

Corollary 12. *We consider two mode functions $S(t)$ and $\overline{S}(t)$ defined for $t \in [0, \tau)$. For given $\delta > 0, \sigma \geq t_m, \sigma + 2\delta < \tau$, the two mode functions differ as follows:*

- *$\overline{S}(t) = 1$ for all $t \in [\sigma, \sigma + \delta)$ ('active mode'),*
- *$\overline{S}(t) = 0$ for all $t \in [\sigma + \delta, \sigma + 2\delta)$ ('idle mode'), and*
- *$S(t) = 1 - \overline{S}(t)$ for all $t \in [\sigma, \sigma + 2\delta)$.*

Then the temperature, denoted as $\overline{T}(\tau)$ at time τ for mode function $\overline{S}(t)$ is not less than the temperature, denoted as $T(\tau)$ at time τ for mode function $S(t)$:

$$\overline{T}(\tau) \geq T(\tau). \tag{10.47}$$

Lemma 13. *For any given time instance τ, we consider two accumulated computing time functions Q, resulting from mode function $S(t)$, and $\overline{Q}$, resulting from mode function $\overline{S}(t)$, with*

$$\overline{Q}(\tau - \Delta, \tau) \geq Q(\tau - \Delta, \tau) \tag{10.48}$$

for all $0 \leq \Delta \leq \tau$ and

$$S(t) = \overline{S}(t) \tag{10.49}$$

for all $t_m < t \leq \tau$. Then, the temperature $\overline{T}(\tau)$ at time τ for mode function $\overline{S}(t)$ is not less than the temperature $T(\tau)$ at time τ for mode function $S(t)$.

Proof. First note that condition (10.48) is equivalent to

$$\overline{Q}(t_m - \Delta, t_m) \geq Q(t_m - \Delta, t_m) \tag{10.50}$$

as $S(t) = \overline{S}(t)$ for all $t_m < t \leq \tau$, and therefore, because of (10.19), the condition can be translated equivalently to

$$\int_{t_m-\Delta}^{t_m} \overline{S}(t)\,\mathrm{d}t \geq \int_{t_m-\Delta}^{t_m} S(t)\,\mathrm{d}t. \tag{10.51}$$

Then, the proof follows from Lemma 2 of [15] by using Lemma 11 and replacing τ with t_m. □

Corollary 14. *For any given time instance τ, we consider two accumulated computing time functions Q, resulting from mode function $S(t)$, and $\overline{Q}$, resulting from mode function $\overline{S}(t)$, with*

$$\overline{Q}(0, \Delta) \geq Q(0, \Delta) \tag{10.52}$$

for all $0 \leq \Delta \leq \tau$ and

$$S(t) = \overline{S}(t) \tag{10.53}$$

for all $0 \leq t < t_m$. Then, the temperature $\overline{T}(\tau)$ at time τ for mode function $\overline{S}(t)$ is not less than the temperature $T(\tau)$ at time τ for mode function $S(t)$.

10.2.3 The Initial Temperature

So far, we assumed that the initial temperature of the system can be neglected and we justified this assumption by the fact that the worst-case computing time is independent of the initial temperature. However, the calculation of a transient upper bound of the temperature requires the consideration of the initial temperature. In the following, we show that the initial temperature has no influence on the highest-temperature at time τ whenever $\mathbf{T}(0) \leq (\mathbf{T}_\infty)^i$, where $(\mathbf{T}_\infty)^i$ is the steady-state temperature of the 'idle' mode.

We know that the output of a linear system results from two different causes, namely the zero-input response and the zero-state response. The zero-state response is the inverse Laplace transformation of the transfer function multiplied by the Laplace transformation of the input. However, as the input $\mathbf{u}$ is assumed to be zero, the zero-input response of a linear system can be calculated by solving the differential equation

$$\frac{\mathrm{d}\mathbf{T}(t)}{\mathrm{d}t} = \mathbf{A} \cdot \mathbf{T}(t) \tag{10.54}$$

and a closed-form solution to the above differential equation yields

$$\mathbf{T}_{\text{zero-input}}(t) = e^{\mathbf{A}\cdot t} \cdot \mathbf{T}(t_0). \tag{10.55}$$

Therefore, without neglecting the initial conditions, the temperature vector of a system can be expressed by

$$\begin{aligned} \mathbf{T}(t) &= \mathbf{T}_{\text{zero-input}}(t) + \mathbf{T}_{\text{zero-state}}(t) \\ &= e^{\mathbf{A}\cdot t} \cdot \mathbf{T}(t_0) + \mathcal{L}^{-1}\left(\mathbf{H}(s) \cdot \mathbf{U}(s)\right) \end{aligned} \tag{10.56}$$

where $\mathbf{H}(\mathbf{s})$ is the Laplace transformation of the impulse response and $\mathbf{U}(\mathbf{s})$ the Laplace transformation of the input.

Theorem 15. *Suppose that the accumulated computing time function $Q^*(0, \Delta)$ for all $0 \leq \Delta \leq \tau$ defined by Theorem 9 leads to temperature $T^*(\tau)$. If, in addition, $\mathbf{T}(0) \leq (\mathbf{T}_\infty)^i$ holds for the initial temperature vector, then for any feasible workload trace we have $T^*(\tau) \geq T(t)$ for all $0 \leq t \leq \tau$.*

Proof. From the definition of the steady-state it follows that applying the 'idle' power as input to the system does not change its temperature as long as the initial temperature of the system is the steady-state temperature of its 'idle' mode. From Lemma 10, we know that the impulse response $h_{ij}(t) \geq 0$ for all $t > 0$ and i, j, and therefore, from (10.38), it follows that the zero-state response caused by any accumulated computing time function Q is never smaller then the zero-state response of a system that only runs in 'idle' mode. From $\mathbf{T}(0) \leq (\mathbf{T}_\infty)^i$ follows that temperatures which are caused by Q satisfy $\mathbf{T}(t) \geq \mathbf{T}(0)$ for all times $0 \leq t \leq \tau$. This holds in particular also for T^*, that is, $T^*(t) \geq T^*(0)$ for all times $0 \leq t \leq \tau$.

Assume for contradiction that $T(\sigma) > T^*(\tau)$ for $\sigma \leq \tau$. We know from Theorem 9 that $T(\sigma)$ is maximized if Q maximizes the optimization function $O(S, \sigma)$ defined in (10.36). However, by the linearity of the system, the mode function $S(t - (\tau - \sigma)$ results in $T(\tau) = T(\sigma)$ for $\mathbf{T}(0) = (\mathbf{T}_\infty)^i$ and

$T(\tau) > T(\sigma)$ for $\mathbf{T}(0) < (\mathbf{T}_\infty)^i$. Consequently, the fact that $T(\sigma) > T^*(\tau)$ would require that $T_k(0) > T_k^*(\tau - \sigma)$ for certain $k \in [1, n]$. As $T^*(t) \geq T^*(0)$ for all times $0 \leq t \leq \tau$, this holds in particular for the time $t = \tau - \sigma$, and therefore, $T^*(\tau - \sigma) \geq T^*(0)$. Having the same initial conditions for both scenarios, that is, $T(0) = T^*(0)$, we find $T(0) \leq T^*(\tau - \sigma)$, which is a contradiction. $\square$

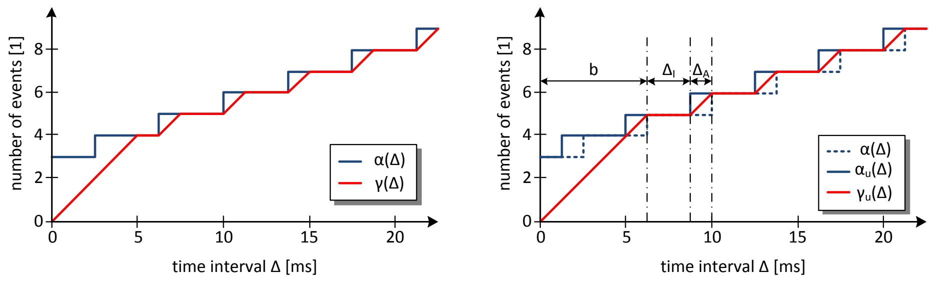
10.3 Period-Jitter-Delay Model

Obviously, there exists no general solution for the optimization problem described in Theorem 9 without using exhaustive search. As every implementation of $S(t)$ on the right side of t_m, that is, for $t > t_m$, directly influences its implementation on the left side of t_m, that is, for $t < t_m$, and vinca versa, no worst-case computing time function can be determined a priori. However, to illustrate the importance of the optimization algorithm, we restrict ourselves to event streams described by a simplified period-jitter-delay model. At first, we create a concrete algorithm that constructs the accumulated computing time $Q^*(0, \Delta)$ for all $0 \leq \Delta \leq \tau$, that leads to a tight upper bound on the highest temperature τ. Afterwards, we determine the corresponding feasible workload trace R^* that leads to the worst-case computing time function Q^*.

10.3.1 Upper Bounded Period-Jitter-Delay Model

Definition 16. *Suppose that the cumulative workload trace of an event stream is upper bounded by an arrival curve $\alpha(\Delta)$ described by the period-jitter-delay model [87] and denote the upper bound of the accumulated computing time function of a work-conserving processing element by $\gamma(\Delta)$. An event stream is described by the* upper bounded period-jitter-delay model*, if its arrival curve $\alpha_u(\Delta)$ is defined so that all active intervals of the corresponding accumulated computing time function $\gamma_u(\Delta)$ have the same length, all idle intervals of $\gamma_u(\Delta)$ have the same length, and $\gamma_u(\Delta) \geq \gamma(\Delta)$. The only exception is the first active interval, that might be longer than all other active intervals.*

The difference between the period-jitter-delay model described in [87] and the upper bounded period-jitter-delay model is also illustrated in Fig. 10.2. In the remaining of this section, we assume that all event streams are described by the upper bounded period-jitter-delay model, and, for notification simplicity, we will not present the index u in the following. Furthermore, we denote the length of the first active interval, also called *burst*, by b, the length of the active intervals by Δ_A, and the length of the idle intervals by Δ_I.



(a) Event stream following the original period-jitter-delay model.

(b) Event stream following the upper bounded period-jitter-delay model.

Figure 10.2: Arrival curve α and the upper bound on the accumulated computing time $\gamma(\Delta)$ for the period-jitter-delay model as described in [87], and the upper bounded period-jitter-delay model to illustrate the difference between these models.

10.3.2 Worst-Case Computing Time

Lemma 17. *Suppose that the event stream is described by the period-jitter-delay model from Definition 16 and $Q^*(0,\Delta)$ for all $0 \leq \Delta \leq \tau$ leads to an upper bound on the highest temperature $T^*(\tau)$ at time τ. Then, the following statements hold:*

- *The processing component is at least $b - \Delta_A$ time units active in one block, that is, $S(t) = 1$ for all $t_l \leq t \leq t_r$ with $t_r - t_l = b - \Delta_A$.*
- *Suppose that the time when $h(\tau - t)$ has its maximum value is denoted by t_m. Then t_m is always between t_l and t_r, that is, $t_l \leq t_m \leq t_r$.*

Proof. Suppose that the accumulated computing time function $Q(0,\Delta)$ for all $0 \leq \Delta \leq \tau$ leads to temperature $T(\tau)$ at time τ and the accumulated computing time function $\overline{Q}(0,\Delta)$ leads to a higher temperature $\overline{T}(\tau)$. We will proof this lemma by showing that every Q, whose processing component does not fulfill both conditions, can be transformed into $\overline{Q}$ with $\overline{T}(\tau) \geq T(\tau)$, that fulfills both conditions. In the following, we will stepwise transform Q into $\overline{Q}$ without violating (10.18), and in each iteration the temperature will not decrease because of Lemma 13 and Corollary 14.

Suppose that $S(t)$ is the mode function of $Q(s,t)$ and $\overline{S}(t)$ the mode function of $\overline{Q}(s,t)$. First note that if

$$\gamma(\Delta) - Q(t - \Delta, t) \geq \Delta x \tag{10.57}$$

for all $0 \leq \Delta \leq t$ and $0 \leq t \leq \tau$, we can perform the transformation

$$S_{new} = \begin{cases} S(t + \Delta x) & 0 \leq t \leq t_m - \Delta x \\ 1 & t_m - \Delta x < t \leq t_m \\ S(t) & t_m < t \leq \tau \end{cases} \tag{10.58}$$

or

$$S_{new} = \begin{cases} S(t) & 0 \leq t \leq t_m \\ 1 & t_m < t \leq t_m + \Delta x \\ S(t - \Delta x) & t_m + \Delta x < t \leq \tau \end{cases} \tag{10.59}$$

to obtain a mode function S_{new} that leads to a higher temperature than mode function S. By Lemma 13 and Corollary 14, respectively, we know that both of these transformations do not reduce the temperature at time τ. Furthermore, (10.18) is not violated as Δx is selected in (10.57) as the maximum amount of time units that can be added without violating $\gamma(\Delta)$. In the following, we assume that the accumulated computing time function $Q(0, \Delta)$ is tight, that is, $\Delta x = 0$.

Now, we will stepwise transform $S(t)$ into $\overline{S}(t)$ by considering the accumulated workload R_p that is upper bounded by an upper arrival curve α_p, where

$$R_p(s, t) \leq \alpha_p(t - s) \quad \forall s < t \tag{10.60}$$

with $\alpha_p(0) = 0$. α_p is described by a periodic event stream with the same period as α, that is, the arrival curve leading to γ, but without any jitter. For a work-conserving schedule, the corresponding accumulated computing time function $Q_p(s, t)$ is upper bounded on the accumulated computing time $\omega(\Delta)$ for intervals with length Δ, that is

$$Q_p(t - \Delta, t) \leq \omega(\Delta) = \inf_{0 \leq \gamma \leq \Delta} \{(\Delta - \lambda) + \alpha_p(\lambda)\}. \tag{10.61}$$

Note that γ is connected with ω as follows:

$$\gamma(\Delta + b - \Delta_A) - \omega(\Delta) = b - \Delta_A \tag{10.62}$$

for all $0 \leq \Delta \leq \tau - b + \Delta_A$. In order to simplify the proof technicalities, we suppose discrete time, that is, $S(t)$ and $\overline{S}(t)$ may change values only at multiples of δ and are constant for $t \in [k\delta, (k + 1)\delta)$ for all $k \geq 0$ and let us define $t_m = k_m\delta$. We split up the mode function $S(t)$ in two parts $S_l(t)$ and $S_r(t)$ with $S(t) = S_l(t) + S_r(t)$ and $S_l(t) = 0$ for all $t \notin [0 \leq t \leq t_m]$ and $S_r(t) = 0$ for all $t \notin (t_m < t \leq \tau]$. Now, we execute the following algorithm:

1. Determine the smallest $1 \leq k_1 \leq k_m$ such that $S(k_1\delta)$ violates $\omega(\Delta)$ when ignoring $S(t)$ for $t > k_1\delta$, that is, there exists a $\Delta > 0$ such that $Q(k_1\delta - \Delta, k_1\delta) > \omega(\Delta)$. If there is no such k_1, then $S_l(t) = \overline{S}_l(t)$ and the algorithm stops.

2. Determine the smallest k_2 with $k_1 < k_2 \leq k_m$ such that $S_l(k_2\delta) = 0$. If such a k_2 exists, then change $S_l(k_1\delta)$ from 1 to 0 and $S_l(k_2\delta)$ from 0 to 1. Otherwise, the algorithm stops.

With the exception of the iteration where the algorithm stops, the temperature $T(\tau)$ increases in every iteration according to Lemma 11. From (10.62) follows, that one can switch at most $b - \Delta_A$ time units in $\omega(\Delta)$ from inactive to active without violating $\gamma(\Delta)$, however, the actual positions of the time units is irrelevant. Therefore, as the algorithm only changes the positions of these excessive active time units, it never violates (10.18). A similar algorithm can be performed on the right side of t_m and by (10.62), there exists a $t_l \leq t_m$ and a $t_r \geq t_m$ so that $t_r - t_l = b - \Delta_A$, and $S(t)$ is always 'active' in this interval. □

Lemma 18. *Suppose that the processing component is continuously active according to Lemma 17 and $S(t)$ is the mode function of an accumulated computing time function $Q(0,\Delta)$ for all $0 \leq \Delta \leq \tau$. $S(t)$ is divided into two parts $S_l(t)$ and $S_r(t)$ with $S(t) = S_l(t) + S_r(t)$ and $S_l(t) = 0$ for all $t \notin [0 \leq t \leq t_m]$ and $S_r(t) = 0$ for all $t \notin (t_m < t \leq \tau]$. Suppose that $S_r(t)$ is selected to maximize the temperature at time τ. Then, the implementation of $S_l(t)$ is only restricted by the positions of the active and idle time intervals in $[t_r, t_r + \Delta_I + \Delta_A)$, and vice versa.*

Proof. Without loss of generality, in the following, we will proof the statement for a given $S_r(t)$. As $t_r - t_l = b - \Delta_A$, the processing component can be at most Δ_A time units active in $[t_r, t_r + \Delta_I + \Delta_A)$ without violating $\gamma(\Delta)$. Now, we will stepwise construct $S_l(t)$ in consideration of $S_r(t)$.

At first, we show that the total amount of active time units of $S_r(t)$ in the interval $[t_r, t_r + \Delta_I + \Delta_A)$ has no influence on $S_l(t)$. However, this follows directly from the fact that $S_l(t)$ cannot have more than Δ_A active time units in $[t_l - \Delta_I - \Delta_A, t_l]$ and $t_r - t_l = b - \Delta_A$. Now, we will show that $S_l(t)$ only depends on the implementation of $S_r(t)$ in $[t_r, t_r + \Delta_I + \Delta_A)$. Suppose that in the time interval $[t_r, t_r + \Delta_I + \Delta_A)$, $S_r(t)$ is a first active at $t_{r,1}$ and is a last active at $t_{r,2}$. Then, without violating $\gamma(\Delta)$, the latest time $S_l(t)$ can be active before t_l is $t_{r,2} - b - \Delta_I$, and cannot be active more than Δ_A between $[t_{r,1} - b - \Delta_I, t_{r,2} - b - \Delta_I)$, and consequently between $[t_{r,1} - b - 2 \cdot \Delta_I, t_{r,2} - b - \Delta_I)$. Figure 10.3 summarizes the most important ideas. To not violate the upper bound of the accumulated computing time, $S_r(t)$ must not have an active trace earlier than $t_{r,1} + \Delta_A + \Delta_I$. However, as

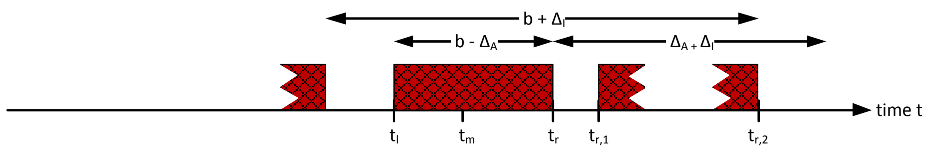


Figure 10.3: Processing time of a component that illustrates the proof of Lemma 18.

the distance to $t_{r,1}$ is at least $\Delta_A + \Delta_I$, it has no influence on $S_l(t)$ anymore, and we can conclude the proof. □

Theorem 19. *Suppose that the event arrival follows the upper bounded period-jitter-delay model from Definition 16. When the scheduler is work-conserving, the following statements hold:*

- *Suppose that the accumulated computing time function $Q^*(0, \Delta)$ calculated by Algorithm 10.1 leads to temperature $T^*(\tau)$ at time τ. Then T^* is an upper bound on the highest temperature at time τ, that is, $T^*(\tau) \geq T(\tau)$.*
- *If, in addition, $\mathbf{T}(0) \leq (\mathbf{T}_\infty)^i$ holds for the initial temperature vector, then for any feasible workload trace we have $T^*(\tau) \geq T(t)$ for all $0 \leq t \leq \tau$.*

Proof. At first, we show that $Q^*(0, \Delta)$ constructed by Algorithm 10.1 satisfies (10.18). Note that, by construction, $Q^*(0, \Delta)$ does not violate (10.18) on the right side of t_r, that is, for $t > t_r$, as there are at most Δ_A active time units in every $\Delta_A + \Delta_I$ interval. Consequently, on the right side of $t_r - b + \Delta_A$, (10.18) is not violated, as well. Next, suppose that V is a possible distribution of Δ_A in $[t_r, t_r + \Delta_A + \Delta_I]$. Whenever V is copied to $[t_l - \Delta_A - \Delta_I, t_l]$, there is always at most b time units active in $b + \Delta_I$ time units, as $t_r - t_l = b - \Delta_A$. As there are no more than Δ_A active time units in every $\Delta_A + \Delta_I$ interval on the left side of $t_r - b + \Delta_A$, the first item is proven.

Now, let us prove that $Q^*(0, \Delta)$ actually lead to an upper bound on the worst-case peak temperature. From Lemma 17 follows that the processing component has to be active at least $b - \Delta_A$ time units, and t_m has to be included in this interval. Lemma 13 and Corollary 14 state that the processing component should be active at most around t_m and finally, Lemma 18 shows that the mode function can be split in two parts S_l and S_r and defines the connection between S_l and S_r. By construction, our algorithm fulfills the condition from Lemma 17. Lemma 13 and Corollary 14 are fulfilled as the same pattern is repeated on both sides of t_m, and the pattern has Δ_A active

Algorithm 10.1: Calculation of the accumulated computing time function $Q^*(0, \Delta)$ for all $0 \leq \Delta \leq \tau$ that leads to the worst-case peak temperature at time τ under the conditions stated in Theorem 19.

```
1: $h(t - \tau) = 0$ for all $\tau < 0$ and $\tau > t$
2: $M = 0$
3: for all $t_r$ in $[t_m, t_m + b - \Delta_A]$ do
4:   for all distributions $V$ of $\Delta_A$ in $[t_r, t_r + \Delta_A + \Delta_I]$ do
5:     set $S$ active in $[t_r - b + \Delta_A, t_r)$
6:     set $S$ active in $V$, starting at $t_r$ and periodically repeated until $\tau$
       with period $\Delta_I + \Delta_A$
7:     set $S$ active in $V$, starting at $t_r - b + \Delta_A$ and periodically repeated
       until 0 with period $\Delta_I + \Delta_A$
8:     $M = (S * h)(t) = \int_0^t S(\tau) \cdot h(t - \tau)\,\mathrm{d}\tau$
9:     if $M > M^*$ then
10:      $M^* = M$
11:      $S^* = S$
12:    end if
13:  end for
14: end for
15: $Q^*(0, \Delta) = \int_0^\Delta S^*(\tau)\,\mathrm{d}\tau$
```

time units per $\Delta_A + \Delta_I$ time units. Only two parameters are left over: the position of the active interval around t_m and the actual distribution of idle and active intervals inside the pattern. Therefore, the worst-case computing time function can be calculated by performing an exhaustive search over these two parameters.

Finally, note that the second item of the theorem is a direct consequence of Theorem 15. □

10.3.3 Worst-Case Accumulated Workload

Theorem 19 only provides an upper bound $T^*(\tau)$ on the worst-case temperature, however, there might be no workload trace that actually leads to the critical accumulated computing time $Q^*(0, \Delta)$. However, in the following, we will show that $R^*(0, \Delta) = Q^*(0, \Delta)$ actually results in the critical accumulated computing time.

Theorem 20. *Suppose that the accumulated computing time function $Q^*(0, \Delta)$ is defined as in Theorem 19. Then the worst-case continuous workload function $R^*(0, \Delta) = Q^*(0, \Delta)$ for all $0 \leq \Delta \leq \tau$*

- *leads to the accumulated computing time $Q^*(0, \Delta)$,*

- *complies to the arrival curve α,*
- *leads to the highest possible temperature $T^*(\tau) \geq T(t)$ for all $0 \leq t \leq \tau$ for any feasible workload trace with the same initial temperature $\mathbf{T}^*(0) = \mathbf{T}(0) \leq (\mathbf{T}_\infty)^i$.*

Proof. Similar to [15], we first have to proof that

$$Q^*(0,\Delta) = \inf_{0\leq u\leq\Delta} \{(\Delta - u) + Q^*(0,u)\} \tag{10.63}$$

as $R^*(0,\Delta) = Q^*(0,\Delta)$. Obviously, there exists a u' such that $(\Delta - u') + Q^*(0,u') = (\Delta - u') + Q^*(0,\Delta)$, namely $u' = \Delta$. Therefore, we only have to show that

$$(\Delta - u) + Q^*(0,u) \geq Q^*(0,\Delta) \tag{10.64}$$

for all $0 \leq u \leq \Delta$. However, as this condition is equivalent to

$$(\Delta - u) \geq Q^*(0,\Delta) - Q^*(0,u) = Q^*(u,\Delta) \tag{10.65}$$

and as the accumulated processing time in interval $[u,\Delta)$ can not exceed the available time $\Delta - u$, it concludes the proof of the first item.

For the second item, we have to show that $R^*(x,y) \leq \alpha(y-x)$. At first, we note that the condition $R^*(0,\Delta) = Q^*(0,\Delta)$ is equivalent to $R^*(x,y) = Q^*(x,y)$, and therefore, from Theorem 19, $R^*(x,y) \leq \gamma(y-x)$. As $\gamma(\Delta) = \inf_{0\leq\lambda\leq\Delta}\{(\Delta-\lambda)+\alpha(\lambda)\} \leq \alpha(\Delta)$, we have $R^*(x,y) \leq \alpha(y-x)$.

The third item is a direct consequence of Theorem 19. First, we see that R^* leads to the accumulated computing time function Q^* and secondly, Q^* leads to the highest temperature $T^*(\tau) \geq T(t)$ according to Theorem 19. □

10.4 Summary

Calculating the worst-case peak temperature is a crucial task in the design of a multi-processor streaming application when real-time constraints have to be guaranteed. In this chapter, we have extended the approach of [15] to calculate the worst-case peak temperature of a single-node system to work-conserving multi-processor systems. In order to enable the specification of non-determinism in the task arrivals, the accumulated workload arriving of every component is separately characterized by an arrival curve

The presented approach includes the following steps for calculating the worst-case peak temperature of component i. First, the problem is reduced to calculating the maximum temperature of component i given that only component j is active and all other components have been removed. The worst-case temperature of component i is then the sum of all individual maximum temperatures. Afterwards, an optimization problem is presented whose solution is the accumulated computing time function that lead to the maximum temperature of component i given that only component j is active.

A concrete implementation of the optimization problem for event streams described by an upper bounded period-jitter-delay model is illustrated. As all arrival curves can be approximated by a periodic event model with jitter [88], this algorithm enables the computation of an upper bound of the maximum temperature of an arbitrary event stream that can be bounded by an arrival curve. However, as the algorithm still includes an exhaustive search, it is usually not feasible for fast temperature analysis and approximation algorithms are required to calculate the worst-case peak temperature of a multi-processor streaming application. Therefore, in the next chapter, we propose and evaluate two approximation algorithms, which tremendously reduce the time to calculate an upper bound on the maximum temperature.

11 Experimental Results

In the last chapter, a framework to determine the worst-case peak temperature of multi-processor streaming applications has been proposed. Now, a prototype implementation of the framework is applied to three case studies to demonstrate the viability of the proposed approach. In the first case study, the maximum temperature of hundred randomly generated workload traces that comply to an arrival curve is compared with its tight worst-case peak temperature. As the calculation of a tight upper bound is very computing intensive, two approximation algorithms are discussed and evaluated in the second case study. Finally, in the third case study, a picture-in-picture application is used to illustrate the advantages of the framework to design real-time systems with timing and peak temperature guarantees.

The chapter is organized as follows: The experimental setup of the first two case studies is described in Section 11.1. Afterwards, in Section 11.2, the mentioned case studies are carried out and its results are discussed. Finally, a summary concludes the chapter.

11.1 Experimental Setup

In the following, the computational and the thermal model to carry out the first two case studies are introduced.

Computational Model. In this evaluation, we consider two different multi-processor streaming applications that are executed on a virtual platform of four processing elements. The computation load of every processing element is modeled as an independent event stream that follows the upper bounded period-jitter-delay model introduced in Section 10.3. The execution demand, the length of the period, and the length of the jitter are selected to satisfy the requirements of the upper bounded period-jitter-delay model. The actual parameters of both applications are outlined in Table 11.1.

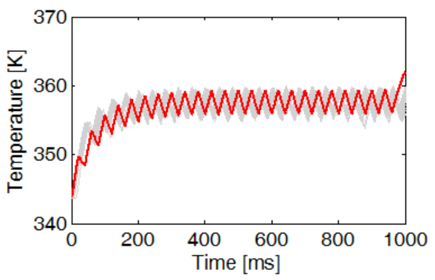
Table 11.1: Parameters of the first and second example application that is used in this section.

		Comp. 1	Comp. 2	Comp. 3	Comp. 4
App. 1	Period p	40 ms	44 ms	24 ms	36 ms
	Jitter j	20 ms	22 ms	48 ms	36 ms
	Min. inter-arrival a	1 ms	1 ms	1 ms	1 ms
	Execution demand d	20 ms	22 ms	12 ms	18 ms
App. 2	Period p	24 ms	36 ms	32 ms	44 ms
	Jitter j	12 ms	36 ms	16 ms	66 ms
	Min. inter-arrival a	1 ms	1 ms	1 ms	1 ms
	Execution demand d	12 ms	18 ms	16 ms	22 ms

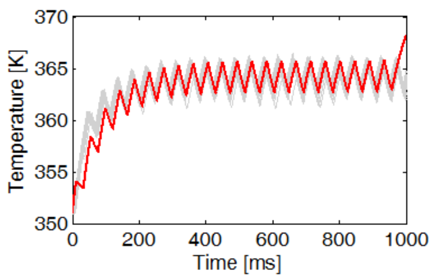
Thermal Model. Unless stated otherwise, a virtual platform of four nodes is used in all evaluations. The thermal and power consumption parameters are outlined in Table 11.2. The parameters for the power consumption are borrowed from [15], and the thermal parameters were set in a way that the steady-state temperatures are inside the normal operating temperature range of modern ICs.

Table 11.2: Power and thermal parameters of the simulated system.

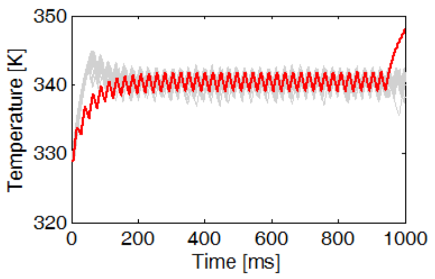
Parameter	Symbol	Value
Thermal conductance [W/K]	g_{ij}	-0.01 to -0.035
Thermal ground conductance [W/K]	s_{jj}	0.65 to 0.75
Thermal capacitance [J/K]	c_{jj}	0.03 to 0.06
Ambient temperature [K]	T_0	300
Slope of power [W/K]	ϕ_{jj}	0.1
Constant power in active mode [W]	ψ_j^a	-11
Constant power in idle mode [W]	ψ_j^i	-25



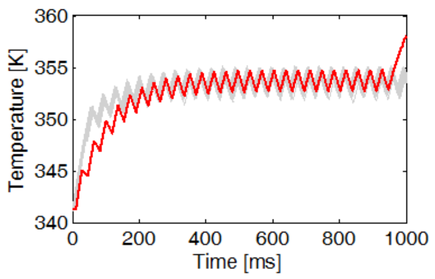
(a) Processing Component 1



(b) Processing Component 2



(c) Processing Component 3



(d) Processing Component 4

Figure 11.1: Transient temperature simulation for measuring the worst-case peak temperature of the *first* example application. The randomly generated workload traces are colored gray and the thermal critical instance is colored red.

11.2 Peak Temperature Analysis

The viability of the proposed framework is analyzed in three case studies. After comparing the tight worst-case peak temperature of an arrival curve with hundred randomly generated workload traces that comply to the same workload curve, two approximation algorithms for the period-jitter-delay model are discussed and evaluated. Finally, in the last case study, the impact of this framework on the design of real-time systems is illustrated by means of a picture-in-picture application.

11.2.1 Comparison with Randomly Generated Workload Traces

The worst-case peak temperature of both applications is first compared with the highest temperature of hundred randomly generated workload traces. In Fig. 11.1, the result of the transient temperature simulation is reported for the first example application. The observation time τ of the worst-case

Table 11.3: The peak temperature, the maximum observed temperature of all random traces, and the average maximum temperature of all randomly generated workload traces for the first and second example application.

		Comp. 1	Comp. 2	Comp. 3	Comp. 4
App. 1	Peak Temp.	361.95 K	368.13 K	347.89 K	357.98 K
	Max. Temp. Random	360.04 K	366.42 K	344.90 K	355.18 K
	Avg. Temp. Random	359.65 K	365.98 K	344.68 K	354.95 K
App. 2	Peak Temp.	360.39 K	369.19 K	344.81 K	359.98 K
	Max. Temp. Random	359.31 K	366.10 K	343.35 K	355.46 K
	Avg. Temp. Random	358.99 K	365.72 K	342.79 K	355.22 K

peak temperature is set to one second and we separately analyze the peak temperature of every component. Therefore, there is one graph for every processing component, which compares 100 randomly generated workload traces that comply to the arrival curves with the thermal critical instance. The thermal critical instance is the worst-case workload that results from Theorem 20. Note that every processing component requires a different set of workload traces to obtain its worst-case peak temperature. A minimal time interval of two milliseconds has been used for the generation of the candidate patterns in the exhaustive search of Algorithm 10.1. All simulations start from the steady-state temperature of the idle mode.

The randomly generated workload traces keep the system constantly at a high temperature, however, all traces under-estimate the worst-case peak temperature. The workload trace that leads to the worst-case peak temperature at time $\tau = 1\,\mathrm{s}$ first warms up the system with periodic arrivals. Around the time, where the corresponding impulse response achieves its maximum value, it heats up the system with burst arrivals and jitters. The peak temperature, the maximum observed temperature of all random traces, and the average maximum temperature of all random traces are summarized in Table 11.3 for both example applications.

11.2.2 Approximation Methods

As the calculation of the tight bound is very expensive in terms of execution time, we investigate two approximation algorithms to calculate an upper bound on the maximum temperature in the second evaluation scenario. Obviously, we are only interested in algorithms, that over-estimate the worst-case peak temperature of a system. In the following, we present two approximation algorithms and compare them to the tight upper bound of the peak temperature calculated by means of Theorem 20.

Algorithm 11.1: Approximation 1:
Computation of the accumulated computing time function $Q^*_{A_1}(0,\Delta)$ for all $0 \leq \Delta \leq \tau$ that leads to an upper bound on the maximum temperature at time τ under the conditions stated in Theorem 19. The resulting temperature is not less than the worst-case peak temperature calculated by means of Theorem 19.

```
1: $h(t-\tau) = 0$ for all $\tau < 0$ and $\tau > t$
2: $M = 0$
3: **for all** $t_r$ in $[tm, t_m + b - \Delta_A]$ **do**
4:     Set $S$ active in $[t_r - b + \Delta_A, t_r)$
5:     Set $S$ active in $[t_r, t_r + \Delta_A)$ and idle in $[t_r + \Delta_A, t_r + \Delta_A + \Delta_I)$, and
       repeat this pattern from $t_r$ to $\tau$
6:     Set $S$ active in $[t_r - b, t_r - b + \Delta_A)$ and idle in $[t_r - b - \Delta_I, t_r - b)$,
       and repeat this pattern from $t_r - b + \Delta_A$ to 0
7:     $M = (S * h)(t) = \int_0^t S(\tau) \cdot h(t-\tau)\,\mathrm{d}\tau$
8:     **if** $M > M^*$ **then**
9:         $M^* = M$
10:        $S^* = S$
11:    **end if**
12: **end for**
13: $Q^*_{A_1}(0,\Delta) = \int_0^\Delta S^*(\tau)\,\mathrm{d}\tau$
```

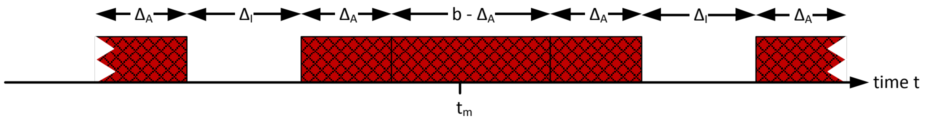
The core idea of *Approximation 1* is to eliminate the exhaustive search over the set of all possible patterns. Therefore, it uses the worst-case pattern with respect to the temperature on both sides of t_m, that is, the time where $h(\tau - t)$ is maximized. The computation of the accumulated computing time function $Q^*_{A_1}(0,\Delta)$ for all $0 \leq \Delta \leq \tau$ that leads to an upper bound on the maximum temperature at time τ is sketched in Algorithm 11.1. *Approximation 2*, that is sketched in Algorithm 11.2, differs from the first one by setting the processing component active for the length of a burst on both sides of t_m. Therefore, it completely eliminates the need of exhaustive search. Figure 11.2 sketches the construction of the accumulated computing time function of *Approximation 1* and *Approximation 2*.

By using the same setup as in the previous evaluation, we compare the worst-case peak temperature calculated by means of Theorem 20 with the maximum temperature calculated by both approximation algorithms. Again, we observe the maximum temperature at time $\tau = 1\,\mathrm{s}$. Table 11.4 summarizes the results for both example applications. The difference between the maximum temperatures calculated by the approximation algorithms and the worst-case peak temperatures is very low, however, the following points need to be considered to interpret the results in detail:

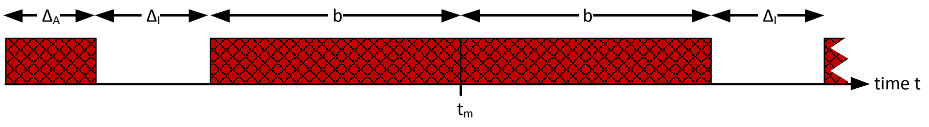
- The power source that has the most influence on the temperature of a node is the one that is connected to this node. However, as the corresponding mirrored impulse response has its maximum at time τ, both approximation algorithms calculate the tight upper bound on the maximum temperature.
- In both example applications, the period and jitter of the arrival curves are selected rather small compared to the characteristic time constants of the thermal model. Therefore, the influence of an additional burst is almost negligible.

Algorithm 11.2: Approximation 2:
Computation of the accumulated computing time function $Q^*_{A_2}(0,\Delta)$ for all $0 \leq \Delta \leq \tau$ that leads to an upper bound on the maximum temperature at time τ under the conditions stated in Theorem 19. The resulting temperature is not less than the temperature calculated by means of Algorithm 11.1.

```
1: $h(t-\tau) = 0$ for all $\tau < 0$ and $\tau > t$
2: $M = 0$
3: Set $S$ active in $[t_m, t_m + b - \Delta_A)$
4: Set $S$ active in $[t_m + b - \Delta_A, t_m + b)$ and idle in $[t_m + b, t_m + b + \Delta_I)$,
   and repeat this pattern from $t_m + b - \Delta_A$ to $\tau$
5: Set $S$ active in $[t_m - b + \Delta_A, t_m)$
6: Set $S$ active in $[t_m - b, t_m - b + \Delta_A)$ and idle in $[t_m - b - \Delta_I, t_m - b)$,
   and repeat this pattern from $t_m - b + \Delta_A$ to 0
7: $Q^*_{A_2}(0,\Delta) = \int_0^\Delta S(\tau)\,\mathrm{d}\tau$
```



(a) Approximation 1



(b) Approximation 2

Figure 11.2: Sketch of the active and inactive intervals of a processing component as calculated by Algorithm 11.1 and Algorithm 11.2, respectively. The length of the burst is denoted by b, the length of an active interval by Δ_A and the length of an inactive interval by Δ_I.

Table 11.4: Comparison of the tight worst-case peak temperature with the maximum temperature calculated by means of *Approximation 1* and *Approximation 2* for the first and second example application.

		Comp. 1	Comp. 2	Comp. 3	Comp. 4
App. 1	Worst-Case Peak T	361.95 K	368.13 K	347.89 K	357.98 K
	Max. T Approx. 1	362.06 K	368.26 K	347.96 K	358.07 K
	Max. T Approx. 2	362.19 K	368.43 K	348.06 K	358.21 K
App. 2	Worst-Case Peak Temp.	360.39 K	369.19 K	344.81 K	359.98 K
	Max. Temp. Approx. 1	360.49 K	369.29 K	344.88 K	360.07 K
	Max. Temp. Approx. 2	360.66 K	369.47 K	344.98 K	360.18 K

Consequently, it can be concluded that both approximation algorithms calculate very accurate estimations of the maximum temperature by a tremendous reduction of the evaluation time.

11.2.3 Picture-In-Picture Application

Motivated by the fact that *Approximation 1* and *Approximation 2* enable fast evaluation of the worst-case peak temperature, we use the proposed approaches to analyze the temperature characteristics of a picture-in-picture application that is mapped onto an architecture of three processing components. A detailed description of the application and the architecture is given in [89], and we only summarize the salient features regarding the temperature analysis in the following.

The picture-in-picture application concurrently decodes two MPEG-2 video streams [63]. The decoding of the video stream takes place in four subsequent actors, that is, the Variable Length Decoding (VLD), the Inverse Quantization (IQ), the Inverse Discrete Cosine Transformation (IDCT), and the Motion Compensation (MC) tasks. Figure 11.3 shows the picture-in-picture application including the considered mapping and architecture. Both of the incoming streams are assumed to have the same resolution and to arrive at the system at a constant bit rate. The architecture consists of three processing components, that are running at different frequencies. PE_1 is running at 1.3 GHz, PE_2 is running at 3 GHz, and PE_3 is running at 1.25 GHz. The input streams and service curves correspond to the ones described in [89].

The thermal parameters shown in Table 11.2 are still valid, but we adjust the power consumption of the processing components to their operational frequency according to (3.7). We assume that the power value ψ_j^a presented in Table 11.2 is valid for an operational frequency of 3 GHz, and that the

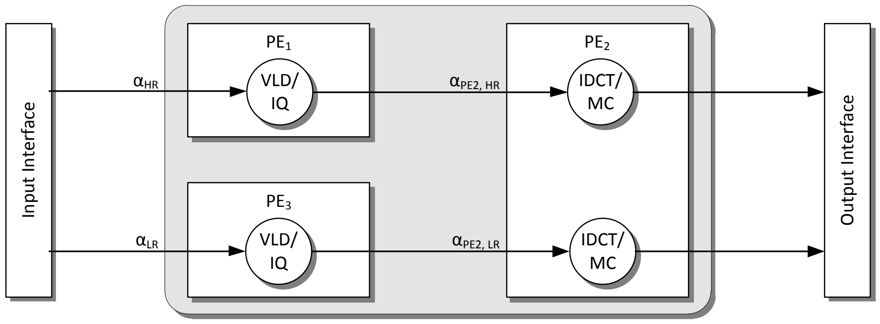


Figure 11.3: Sketch of the picture-in-picture application that decodes two MPEG-2 video streams in parallel. Only the components, which are relevant for the thermal evolution are outlined.

supply voltage is constant. Therefore, we get $\psi_1^a = -18.93\,\mathrm{W}$, $\psi_2^a = -11\,\mathrm{W}$, and $\psi_3^a = -19.16\,\mathrm{mW}$ for the constant power in the active mode.

In order to use the presented approximation algorithms to calculate an upper bound on the worst-case peak temperature of the system, the arrival curves have to be approximated by such that are described by the upper bounded period-jitter-delay model introduced in Section 10.3. Figure 11.4 illustrates the required steps for that approximation based on the arrival curve of pro-

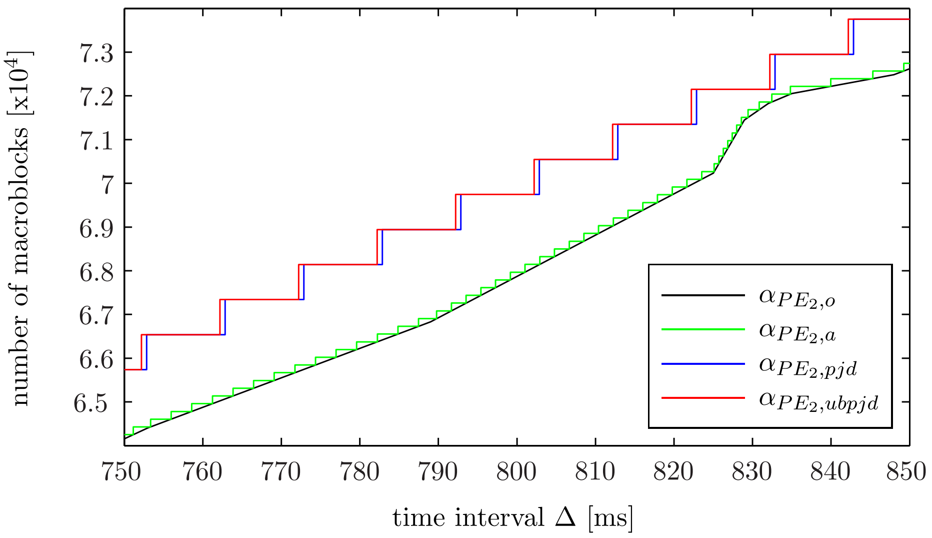
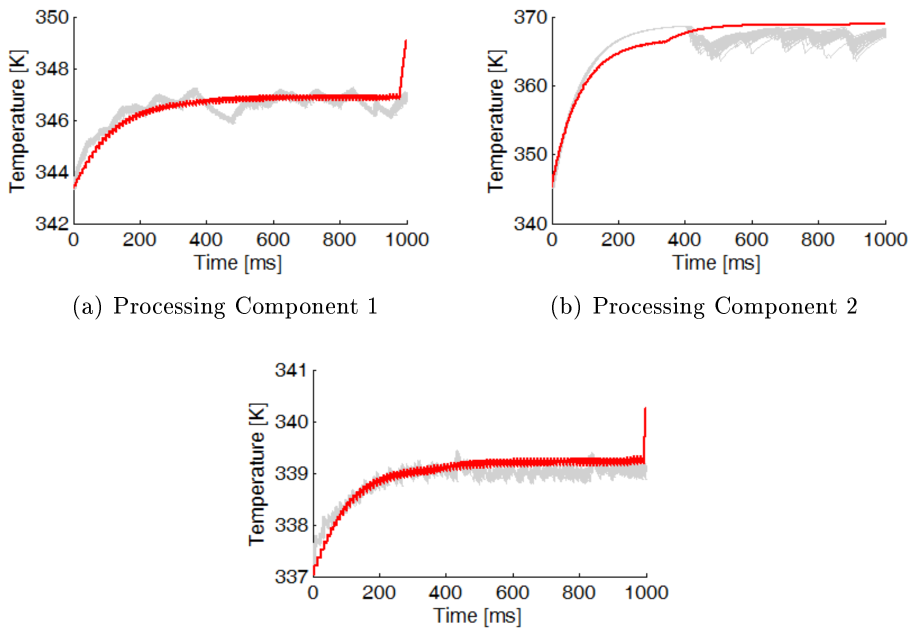


Figure 11.4: Illustration of the steps that are required to approximate a general arrival curve by one that is described by the upper bounded period-jitter-delay model. The arrival curve of processing component 2 serves as example.



(a) Processing Component 1 (b) Processing Component 2

(c) Processing Component 3

Figure 11.5: Transient temperature simulation for measuring the worst-case peak temperature of the picture-in-picture application. The randomly generated workload traces are colored gray and the thermal critical instances are colored red.

cessing component PE_2. The accumulated input stream of processing component PE_2 is bounded by the arrival curve $\alpha_{PE_2,o} = \alpha_{PE_2,LR} + \alpha_{PE_2,HR}$. However, as the arrival curve has time continuous event arrivals, it is first approximated by an arrival curve $\alpha_{PE_2,a}$ that has discrete event arrivals. Afterwards, the approach proposed in [88] is used to approximate $\alpha_{PE_2,a}$ by an arrival curve that is described by a period event model with jitter and we denote the new arrival curve by $\alpha_{PE_2,pjd}$. Finally, the arrival curve $\alpha_{PE_2,pjd}$ is approximated by the upper bounded period-jitter-delay model and the resulting arrival curve is denoted as $\alpha_{PE_2,ubpjd}$.

The worst-case peak temperature is measured at time $\tau = 1\,\mathrm{s}$ and Fig. 11.5 presents the result of the transient temperature simulation for the picture-in-picture application in the interval $[0\,\mathrm{s}, 1\,\mathrm{s}]$. The thermal critical instance, that is, the worst-case workload trace, is compared with 40 randomly generated workload traces that comply to the arrival curve α_{PE_2}. All traces start from the steady-state temperature of its 'idle' mode and the thermal critical instance is calculated by *Approximation 1*. The randomly generated workload traces underestimate the worst-case peak temperature on average by almost one Kelvin. Note that the maximum temperature of the worst-

Table 11.5: The peak temperature, the maximum observed temperature of all random traces, and the average maximum temperature of all random traces of the transient temperature simulation of Fig. 11.5.

	Comp. 1	Comp. 2	Comp. 3
Peak Temp.	349.08 K	368.93 K	340.25 K
Max. Temp. Rand. Traces	347.26 K	368.67 K	339.45 K
Avg. Temp. Rand. Traces	347.23 K	368.66 K	339.41 K

case peak temperature is not any more a tight worst-case peak temperature as the actual arrival curves have been upper bounded and the thermal critical instance has been calculated by means of an approximation algorithm. Table 11.5 summarizes the peak temperatures of the traces shown in Fig. 11.5. As the worst-case peak temperature of processing component 2 is much higher than the one of the other processing components, the workload of processing component 2 leads to a hot spot, and the application designer might try another mapping with a lower worst-case peak temperature.

11.3 Summary

In this chapter, the necessity to determine the worst-case peak temperature of a streaming application is illustrated in three case studies. The scenario used in the first two case studies includes a virtual platform of four nodes, and the event streams of every processing component are modeled as independent arrival curves. The comparison of hundred randomly generated workload traces with the thermal critical instance illustrates that thermal simulation is not suited for determining safe bounds for hard real-time systems.

As exhaustive search is infeasible to compare the thermal characteristics of hundreds of design alternatives, we presented two approximation algorithms that overestimate the worst-case peak temperature. The evaluation shows that both methods provide an accurate approximation of the worst-case peak temperature, and therefore, are feasible alternatives for measuring the maximum temperature of a multi-processor streaming application in an early design stage. Motivated by these results, the worst-case peak temperature of a realistic case study involving a picture-in-picture application is performed. It illustrates that the proposed analytic approach is a cornerstone to design real-time systems that have to guarantee both thermal constraints and real-time deadlines.

12 Analytic Thermal Analysis: Conclusion and Outlook

12.1 Conclusion

Exceeding the worst-case peak temperature can have drastic impacts on the reliability of a system. Therefore, the identification of the worst-case peak temperature is a crucial task in the design of multi-processor applications for embedded real-time systems in order to guarantee that real-time constraints are met. However, as thermal simulation methods only cover a fraction of all possible system behaviors, they cannot be used to determine hard bounds on the temperature and novel analytic approaches are required.

The second part of this thesis discussed analytic worst-case methods to obtain the maximum temperature of a many-core system. To this end, the approach presented in [15] to calculate the worst-case peak temperature of a single-node system has been extended to multi-processor systems. The novel framework calculates a tight upper bound on the worst-case peak temperature of a many-core system for given arrival curves under work-conserving real-time scheduling algorithms. By using the superposition principle, the problem is reduced to determine the maximum temperature of a component given that only one processing component is active and all other components have zero power dissipation. The worst-case peak temperature of a component is the sum of all individual maximum temperatures. Then, the accumulated computing time function that leads to the maximum temperature, can be determined by solving an optimization problem involving the impulse response and a single side condition.

The optimization problem is solved for event streams described by the period-jitter-delay model and its implementation is used to compare the proposed framework with randomly generated workload traces. However, as the solution is too expensive in terms of evaluation time, two approximation algorithms are proposed and evaluated. Both of them tremendously reduce the evaluation time by providing an accurate approximation of the worst-case peak temperature. The impact of the proposed framework on the design of multi-processor real-time systems is fundamental as it gives guarantees on the worst-case temperature of the system, which in turn could avoid the use of dynamic thermal management techniques.

12.2 Outlook

Obviously, the framework to determine the worst-case peak temperature is still in its fledgling stages and there is a huge amount of interesting topics for future research.

From the analytical point of view, it is required to determine conditions on the thermal model, which legitimate our assumptions on the impulse response. Afterwards, efficient approximation methods are required that cover a much wider class of event streams than only period-jitter-delay models. As the thermal model of a typical multi-processor platform includes more than hundred nodes, model reduction is required to reduce the computational overhead in calculating the worst-case peak temperature. Another interesting extension of the framework would be to support more than two power modes. Finally, the task of automatically generating MPA models for the thermal analysis should be investigated in order to integrate this framework into the design space exploration of a multi-processor streaming application.

13 Conclusion

Recently, research studies have shown that mobile devices will replace personal computers by 2014 [90]. However, the current generation of processors for mobile devices is not yet able to deal with the requirements of future high performance applications [91] and a new generation of processors for mobile devices becomes mandatory. Current trends include the design of mobile processors as heterogeneous or homogeneous MPSoCs. However, the increase in performance imposes a major rise in power density, which in turn has a high impact on the reliability of the system. Consequently, it becomes infeasible to guarantee real-time constraints. This master thesis proposed a set of novel techniques to evaluate the thermal characteristics of a many-core embedded system in a very early design stage. More precisely, the proposed techniques include an approach for automated calibration of abstract thermal evaluation models, methods for simulating the transient temperature evolution of a system described by its software synthesis specifications, and an analytic framework to compute the tight worst-case peak temperature of a many-core system.

The selection of a thermal evaluation method for the comparison of design alternatives of a multi-processor application is always a tradeoff between accuracy and evaluation speed. Figure 13.1 illustrates this tradeoff visually by means of the methods derived during this master thesis. In particularly, one can draw the following conclusions regarding the design flow of an embedded real-time system:

The low-level thermal evaluation tool chain is obviously the slowest evaluation method, but provides very high accuracy. It has the advantage that

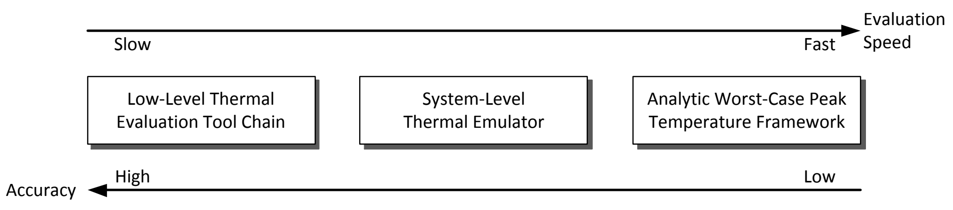

Evaluation Speed
Slow
Fast
Low-Level Thermal Evaluation Tool Chain
System-Level Thermal Emulator
Analytic Worst-Case Peak Temperature Framework
Accuracy
High
Low

Figure 13.1: Tradeoff between accuracy and evaluation speed for the thermal evaluation.

no model calibration is required and that the setup of the tool chain is straightforward. It is mainly used for the support of more abstract models during their calibration and for the verification of the final design in a late stage of the design process.

The System Level Thermal Emulator (SLTE) uses timing and thermal parameters that are collected in advance to improve the evaluation speed. Its area of application is an earlier stage of the design process, where several design alternatives are compared in detail. As SLTE relies on model calibration, it is only faster than the low-level thermal evaluation tool chain when more than one design alternatives are evaluated. In comparison with the current state-of-the-art methods for thermal evaluation, namely FPGA emulation platforms, SLTE achieves similar speed-ups and accuracies, but is much more flexible and cheaper. Its drawbacks include the evaluation speed that decreases with the problem, application and architecture complexity. In comparison with other hardware/software thermal evaluation approaches, it is limited to streaming applications and requires a detailed model calibration to provide accurate results. Although the model calibration is fully automated, it is still time consuming and increases the overall evaluation time in the design space exploration.

As thermal simulation methods only cover a fraction of all possible system behaviors, they cannot be used to determine hard bounds on the temperature as required by hard real-time systems. The third thermal evaluation method derived in this thesis tackles this problem by analytically calculating the worst-case peak temperature of a many-core system. In addition to cover all possible system behaviors, such analytic methods have the advantage of a very high evaluation speed. The proposed framework enables system designers to compare a huge number of design alternatives already in an early design stage. Therefore, the impact of the proposed method on the design of multi-processors is fundamental as it is the first framework that gives guarantees on the worst-case peak temperature of a many-core system. However, before the framework becomes applicable to a wide range of application scenarios, the following limitations have to be addressed: The current framework has a rather limited modeling scope over simulation based evaluation methods. Effects caused by the OS, that is, context or process invocation overhead, are rather hard to implement in an analytic evaluation model, but might have considerable impact on its accuracy. Moreover, the current analytical model only covers two power modes, however, the power dissipation of a real platform often covers a whole range.

This master thesis proposed various techniques to evaluate the temperature behavior of a many-core system that is described by its high-level system specification. We have shown that every method targets another stage in the design process of a multi-processor application and highlighted the importance of an accurate calibration of the underlying models. Therefore, future research has to answer the question of how to obtain accurate power and temperature models, in particularly of real platforms where detailed information about the underlying architecture is often missing.

A
Details of Experimental Results

A.1 Transient Thermal Behavior

In this section, additional information about the experimental settings, and supplementary results for the applications presented in Chapter 7 as well as for an FFT and an IIR filter application are listed.

A.1.1 Mapping Specifications

Tables A.1 to A.5 show the mapping specifications that are used for the experiments in Chapter 7. The mapping is noted as a pair “binding/priority”. The identity number of the processor is used to specify the binding and an unsigned integer is used to specify the priority in which lower numerical values refer to higher priorities.

A.1.2 Details of the Presented Experiments

This subsection shows additional details of the experiments presented in Chapter 7. In Figs. A.1 and A.2, the transient temperature behavior of the second and forth candidate mapping of the evaluation of the matrix multiplication application in Table 7.6 is plotted, respectively. Similarly, in Figs. A.3 and A.4, the transient temperature behavior of the second candidate mapping of the MJPEG decoder in Table 7.8 is plotted and in Figs. A.5 and A.6, the transient temperature of the forth candidate mapping is plotted.

Table A.1: Mappings used in the evaluation of different chain lengths of the producer-consumer application (Table 7.3).

	Producer	Consumer	Worker 1	Worker 2	Worker 3	Worker 4	Worker 5	Worker 6	Worker 7	Worker 8
1 WP	0/1	2/1	1/1	-	-	-	-	-	-	-
2 WPs	0/1	2/1	1/1	1/2	-	-	-	-	-	-
3 WPs	0/1	0/2	1/1	2/1	1/2	-	-	-	-	-
4 WPs	0/1	0/2	1/1	2/1	1/2	2/2	-	-	-	-
5 WPs	0/1	0/2	1/1	2/1	1/2	2/2	1/3	-	-	-
6 WPs	0/1	0/2	1/1	2/1	1/2	2/2	1/3	2/3	-	-
7 WPs	0/1	0/2	1/1	2/1	1/2	2/2	1/3	2/3	1/4	-
8 WPs	0/1	0/2	1/1	2/1	1/2	2/2	1/3	2/3	1/4	2/4

Table A.2: Mappings used in the evaluation of different mappings of the producer-consumer application (Table 7.4).

	Producer	Consumer	Worker 1	Worker 2	Worker 3
Mapping 1	0/1	0/2	1/1	2/1	1/2
Mapping 2	0/1	0/2	1/1	1/2	2/1
Mapping 3	0/1	2/2	0/2	1/1	2/1
Mapping 4	0/1	2/1	0/2	1/1	1/2
Mapping 5	0/1	2/1	1/1	1/2	1/3
Mapping 6	0/1	2/1	1/3	1/2	1/1
Mapping 7	0/2	0/1	1/2	2/1	1/1

Table A.3: Mappings used in the evaluation of different mappings of the matrix multiplication application (Table 7.6).

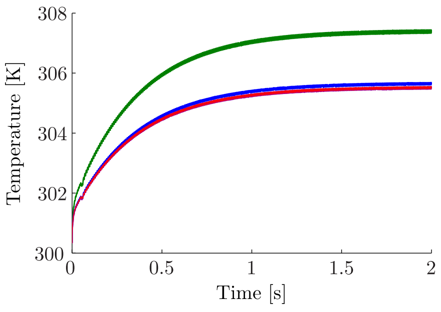
	Producer	Consumer	Mult 0/0/0	Mult 0/0/1	Mult 0/1/0	Mult 0/1/1	Mult 1/0/0	Mult 1/0/1	Mult 1/1/0	Mult 1/1/1
Mapping 1	0/1	0/2	1/1	1/2	1/3	2/2	1/4	2/3	2/1	2/4
Mapping 2	0/1	2/1	0/2	0/3	1/1	1/3	1/2	1/4	2/2	2/3
Mapping 3	0/1	1/1	2/1	2/2	2/3	2/6	2/4	2/7	2/5	2/8
Mapping 4	0/2	0/1	1/4	1/3	1/2	2/2	1/1	2/3	2/4	2/1
Mapping 5	0/3	1/2	0/1	1/3	1/4	0/2	2/1	2/2	2/3	1/1
Mapping 6	0/3	1/3	0/1	0/2	1/2	1/1	2/1	2/2	2/3	2/4
Mapping 7	0/2	0/3	1/1	1/2	1/3	1/4	2/1	2/3	2/2	0/1

Table A.4: Mappings used in the evaluation of different mappings of the MJPEG decoder application when decoding one frame per time (Table 7.7).

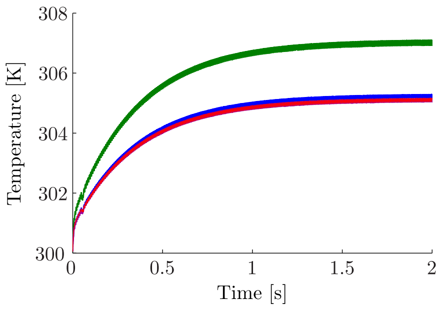
	Splitstream	Splitframe	Decode	Mergeframe	Mergestream
Mapping 1	0/1	1/1	2/1	1/2	0/2
Mapping 2	0/2	1/2	2/1	1/1	0/1
Mapping 3	0/1	1/1	0/3	2/1	0/2
Mapping 4	0/3	1/1	0/1	2/1	0/2

Table A.5: Mappings used in the evaluation of different mappings of the MJPEG decoder application when decoding two frames in parallel (Table 7.8).

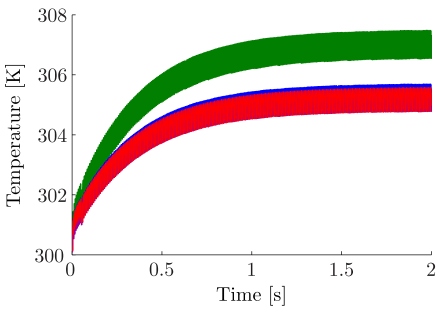
	Splitstream	Splitframe 0	Decode 0	Mergeframe 0	Splitframe 1	Decode 1	Mergeframe 1	Mergestream
Mapping 1	0/1	1/1	2/1	3/1	1/2	2/2	3/2	0/2
Mapping 2	0/2	1/2	2/2	3/2	1/1	2/1	3/1	0/1
Mapping 3	0/1	0/2	1/1	2/1	1/2	2/2	3/1	3/2
Mapping 4	0/1	1/1	2/1	3/1	3/2	1/2	2/2	0/2



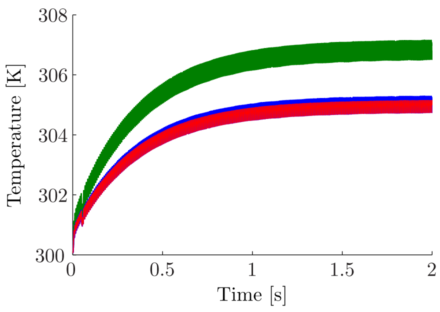
(a) Tile 0, Core



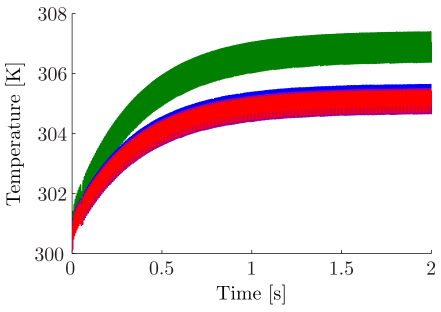
(b) Tile 0, Data Cache.



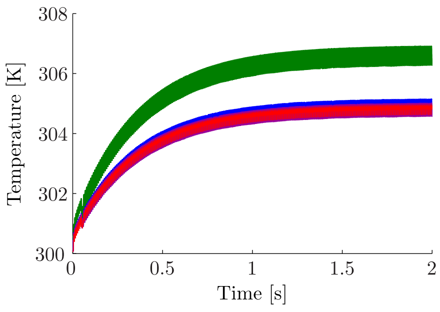
(c) Tile 1, Core



(d) Tile 1, Data Cache



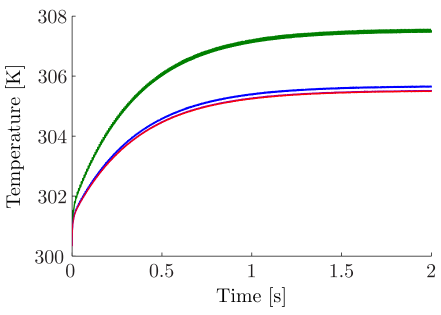
(e) Tile 2, Core



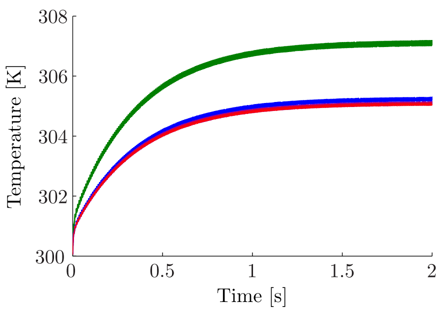
(f) Tile 2, Data Cache

🟩 SLTE: Peak 🟦 SLTE: Max. Average 🟪 SLTE: Average 🟥 Low-Level Simulation

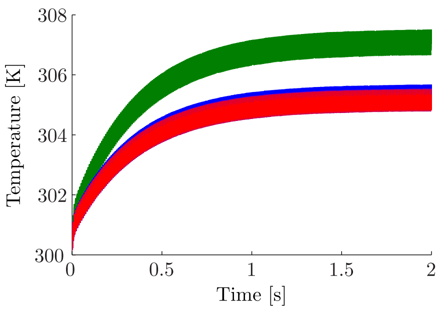
Figure A.1: Matrix Multiplication, Mapping II:
Temperature evolution of three tiles of the MPARM virtual platform when the matrix multiplication application is executed. For each tile, both the core and data cache is outlined. The application has been simulated with SLTE using three different power values, that is, the average, maximum average, and peak power consumption.



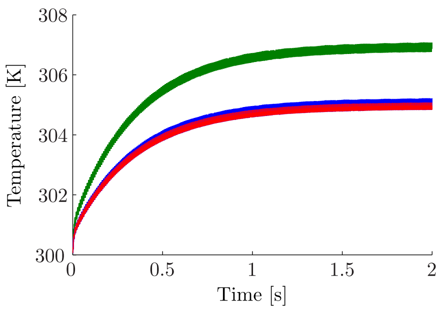
(a) Tile 0, Core



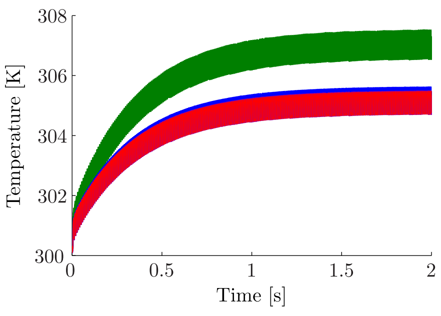
(b) Tile 0, Data Cache



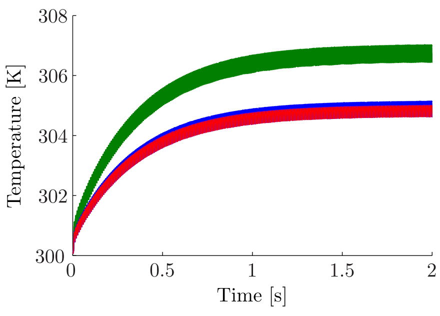
(c) Tile 1, Core



(d) Tile 1, Data Cache

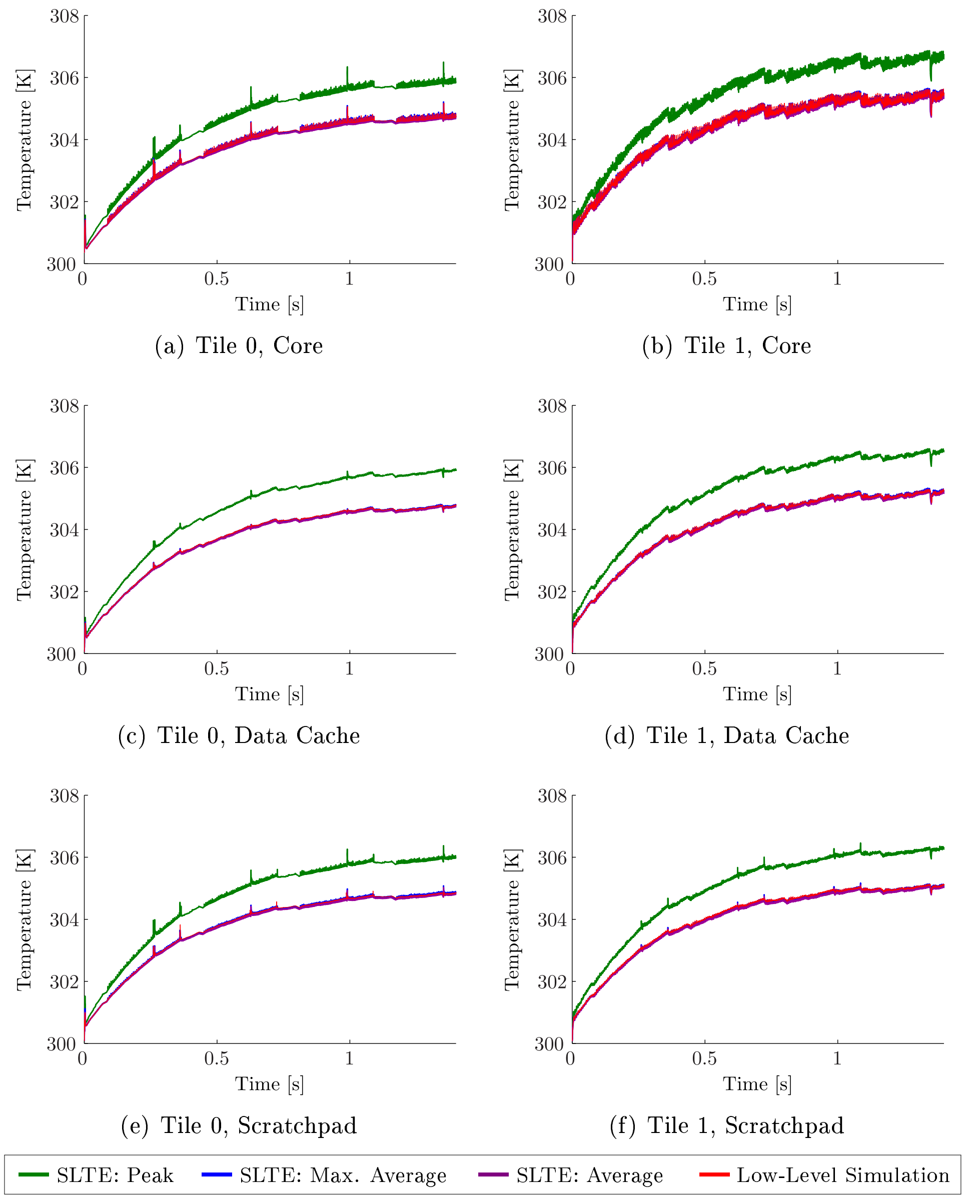


(e) Tile 2, Core



(f) Tile 2, Data Cache

Figure A.2: Matrix Multiplication, Mapping IV:
Temperature evolution of three tiles of the MPARM virtual platform when the matrix multiplication application is executed. For each tile, both the core and data cache is outlined. The application has been simulated with SLTE using three different power values, that is, the average, maximum average, and peak power consumption.



(a) Tile 0, Core

(b) Tile 1, Core

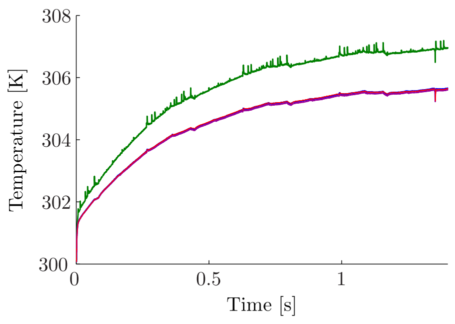
(c) Tile 0, Data Cache

(d) Tile 1, Data Cache

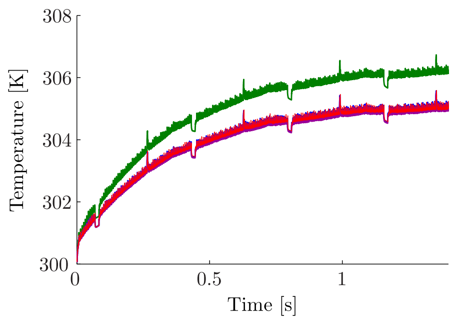
(e) Tile 0, Scratchpad

(f) Tile 1, Scratchpad

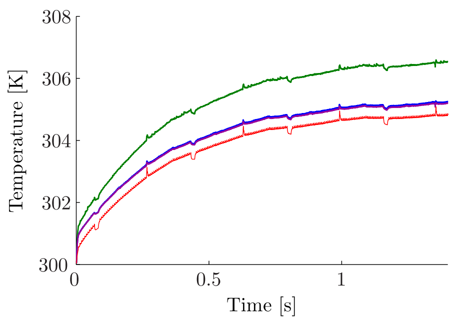
Figure A.3: MJPEG Decoder, Mapping II, Tile 1 & 2:
Temperature evolution of two tiles of the MPARM virtual platform when the MJPEG decoder application is executed. Three components, namely the core, data cache and scratchpad, are outlined for each tile. The application has been simulated with SLTE using three different power values, that is, the average, maximum average, and peak power consumption.



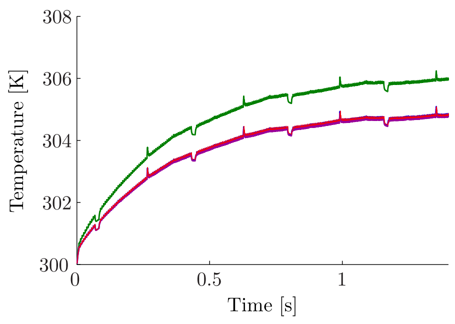
(a) Tile 2, Core



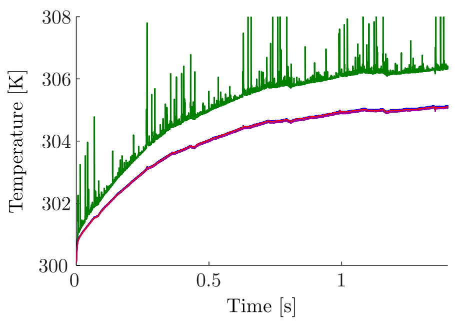
(b) Tile 3, Core



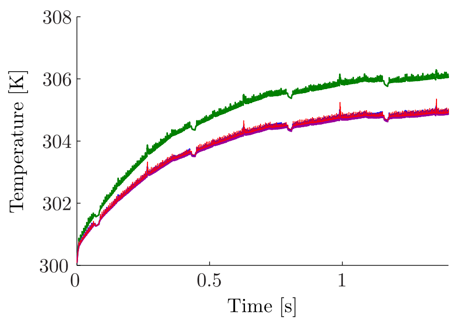
(c) Tile 2, Data Cache



(d) Tile 3, Data Cache



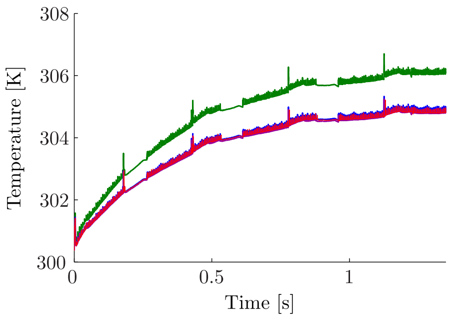
(e) Tile 2, Scratchpad



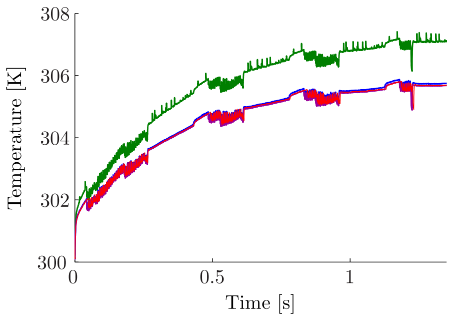
(f) Tile 3, Scratchpad

🟩 SLTE: Peak 🟦 SLTE: Max. Average 🟪 SLTE: Average 🟥 Low-Level Simulation

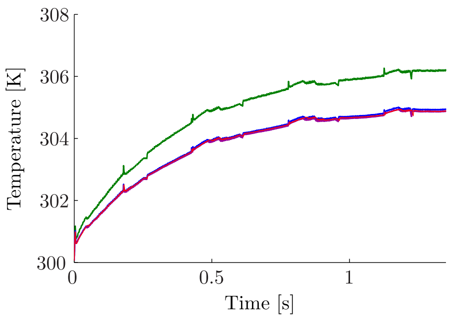
Figure A.4: MJPEG Decoder, Mapping II, Tile 3 & 4:
Temperature evolution of two tiles of the MPARM virtual platform when the MJPEG decoder application is executed. Three components, namely the core, data cache and scratchpad, are outlined for each tile. The application has been simulated with SLTE using three different power values, that is, the average, maximum average, and peak power consumption.



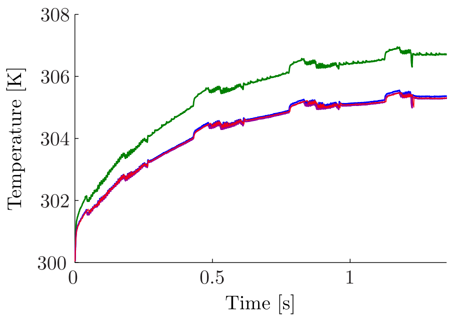
(a) Tile 0, Core



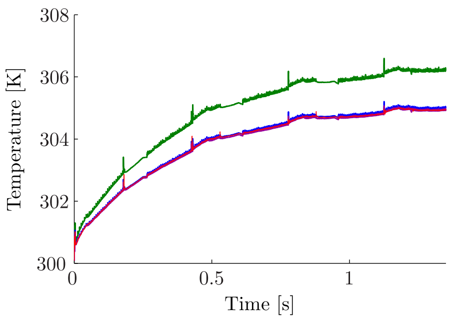
(b) Tile 1, Core



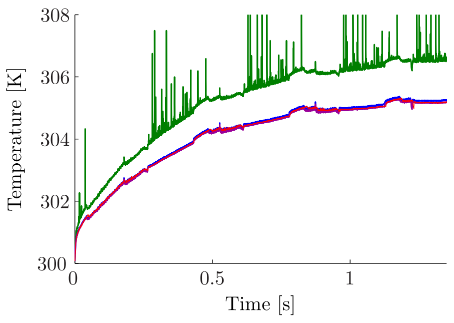
(c) Tile 0, Data Cache



(d) Tile 1, Data Cache



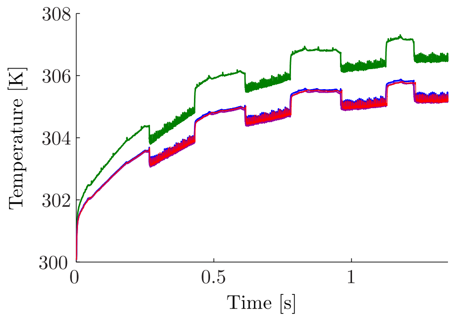
(e) Tile 0, Scratchpad



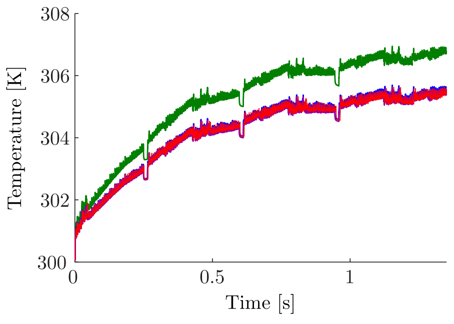
(f) Tile 1, Scratchpad

🟩 SLTE: Peak 🟦 SLTE: Max. Average 🟪 SLTE: Average 🟥 Low-Level Simulation

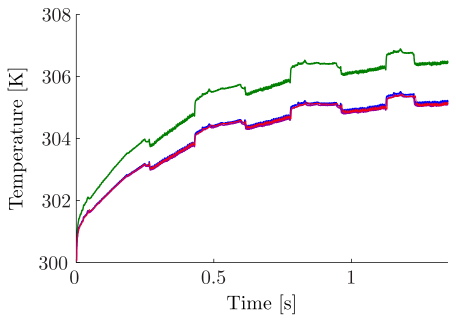
Figure A.5: MJPEG Decoder, Mapping IV, Tile 1 & 2:
Temperature evolution of two tiles of the MPARM virtual platform when the MJPEG decoder application is executed. Three components, namely the core, data cache and scratchpad, are outlined for each tile. The application has been simulated with SLTE using three different power values, that is, the average, maximum average, and peak power consumption.



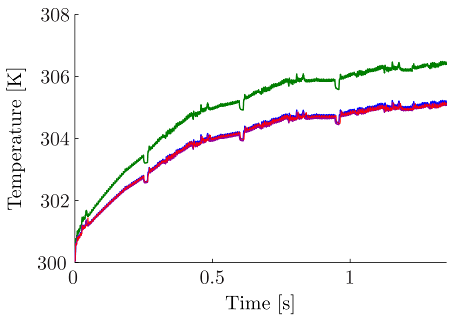
(a) Tile 2, Core



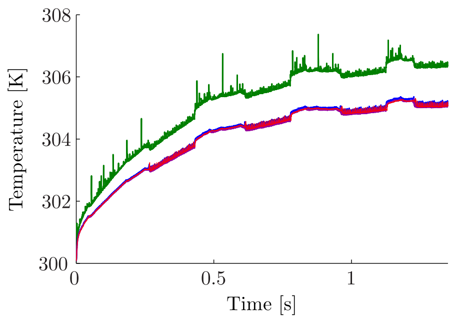
(b) Tile 3, Core



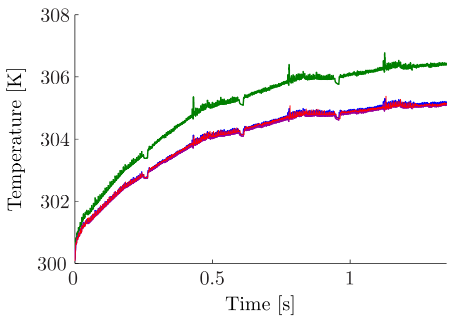
(c) Tile 2, Data Cache



(d) Tile 3, Data Cache



(e) Tile 2, Scratchpad



(f) Tile 3, Scratchpad

🟩 SLTE: Peak 🟦 SLTE: Max. Average 🟪 SLTE: Average 🟥 Low-Level Simulation

Figure A.6: MJPEG Decoder, Mapping IV, Tile 3 & 4:
Temperature evolution of two tiles of the MPARM virtual platform when the MJPEG decoder application is executed. Three components, namely the core, data cache and scratchpad, are outlined for each tile. The application has been simulated with SLTE using three different power values, that is, the average, maximum average, and peak power consumption.

A.1.3 Additional Experiments

In the following, supplementary results for an IIR filter and an FFT application are listed. The reader is referred to [92] for a detailed description of both applications.

Infinite Impulse Response Filter. The evaluation of four candidate mappings of a first order IIR filter application is considered next. The first order IIR filter can be described by its state equation as

$$y[n] = x[n] + c \cdot y[n-1] \tag{A.1}$$

where x denotes the sequences of input samples, y the sequence of output samples, and c a constant. The results of the evaluation are reported in Table A.6 and the mapping specifications are shown in Table A.7.

Table A.6: Infinite Impulse Response Filter, Different Mappings: Pessimism, execution times and speed-ups of the IIR filter application where four different mappings are compared. The application is mapped onto a distributed architecture of two tiles and the mapping that is used to perform the model calibration is marked with a star.

	Pessimism			**Execution Time**		
	Average	*Min*	*Max*	*Platform*	*SLTE*	*Speed-up*
Mapping 1$*$	0.0038	0.0002	0.0121	3.75 s	2.58 s	2197
Mapping 2	0.0030	0.0001	0.0086	3.75 s	2.51 s	2285
Mapping 3	0.0071	0.0054	0.0102	5.25 s	2.50 s	2726
Mapping 4	0.0087	0.0077	0.0126	4.94 s	2.32 s	2769

Table A.7: Mappings used in the evaluation of different mappings of the first order IIR filter application (Table A.6).

	Producer	Filter	Consumer
Mapping 1	1/2	0/1	1/1
Mapping 2	1/1	0/1	1/2
Mapping 3	0/1	0/2	1/1
Mapping 4	0/1	1/1	1/2

Fast Fourier Transform. In the last evaluation of this master thesis, a distributed implementation of a 2-point FFT application is considered. The results of the evaluation are reported in Table A.8 and the mapping specifications are shown in Table A.9.

Table A.8: Fast Fourier Transform, Different Mappings:
Pessimism, execution times and speed-ups of the FFT application where four different mappings are compared. The application is mapped onto a distributed architecture of three tiles and the mapping that is used to perform the model calibration is marked with a star.

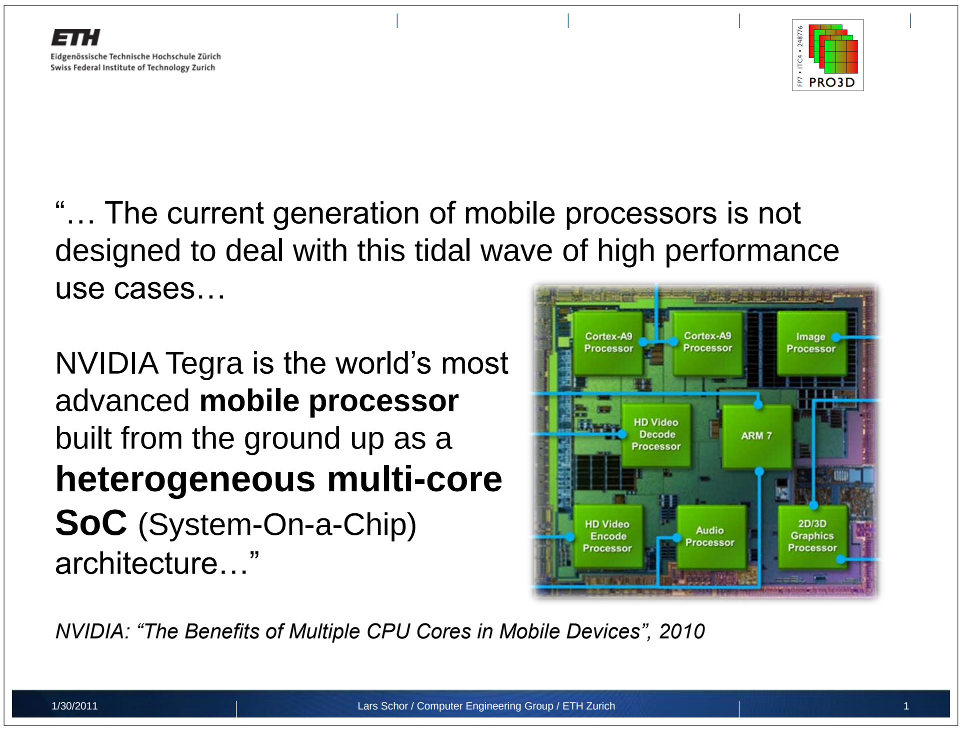
	Pessimism			**Execution Time**		
	Average	*Min*	*Max*	*Platform*	*SLTE*	*Speedup*
Mapping 1∗	0.0056	0.0037	0.0062	1.71 s	3.88 s	751
Mapping 2	0.0063	0.0041	0.0100	3.01 s	2.10 s	2030
Mapping 3	0.0068	0.0049	0.0116	3.01 s	1.97 s	2165
Mapping 4	0.0068	0.0052	0.0082	1.82 s	8.11 s	379

Table A.9: Mappings used in the evaluation of different mappings of the FFT application (Table A.8).

	Generator	Consumer	FFT2_0_0	FFT2_0_1	FFT2_1_0	FFT2_1_1
Mapping 1	0/2	1/1	0/1	1/2	2/2	2/1
Mapping 2	0/1	0/2	1/1	1/2	2/1	2/2
Mapping 3	0/2	0/1	1/2	1/1	2/2	2/1
Mapping 4	0/1	1/2	0/2	1/1	2/1	2/2

B

Presentation Slides


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

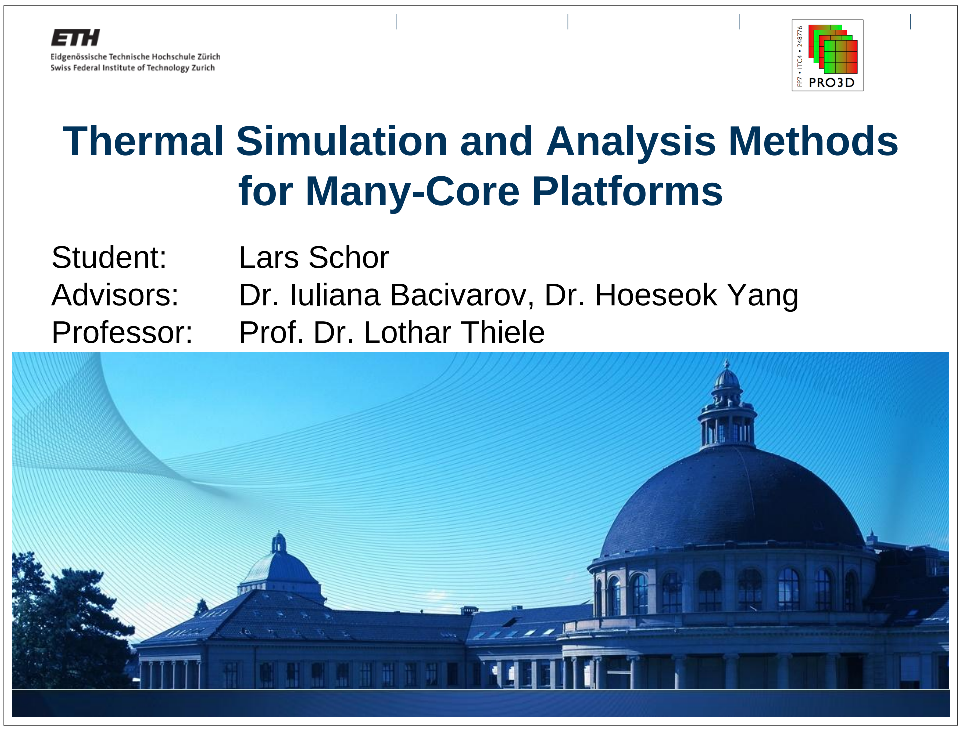
FP7 • ITC4 • 248776
PRO3D

"… The current generation of mobile processors is not designed to deal with this tidal wave of high performance use cases…

NVIDIA Tegra is the world's most advanced **mobile processor** built from the ground up as a **heterogeneous multi-core SoC** (System-On-a-Chip) architecture…"

NVIDIA: "The Benefits of Multiple CPU Cores in Mobile Devices", 2010

1/30/2011 | Lars Schor / Computer Engineering Group / ETH Zurich | 1

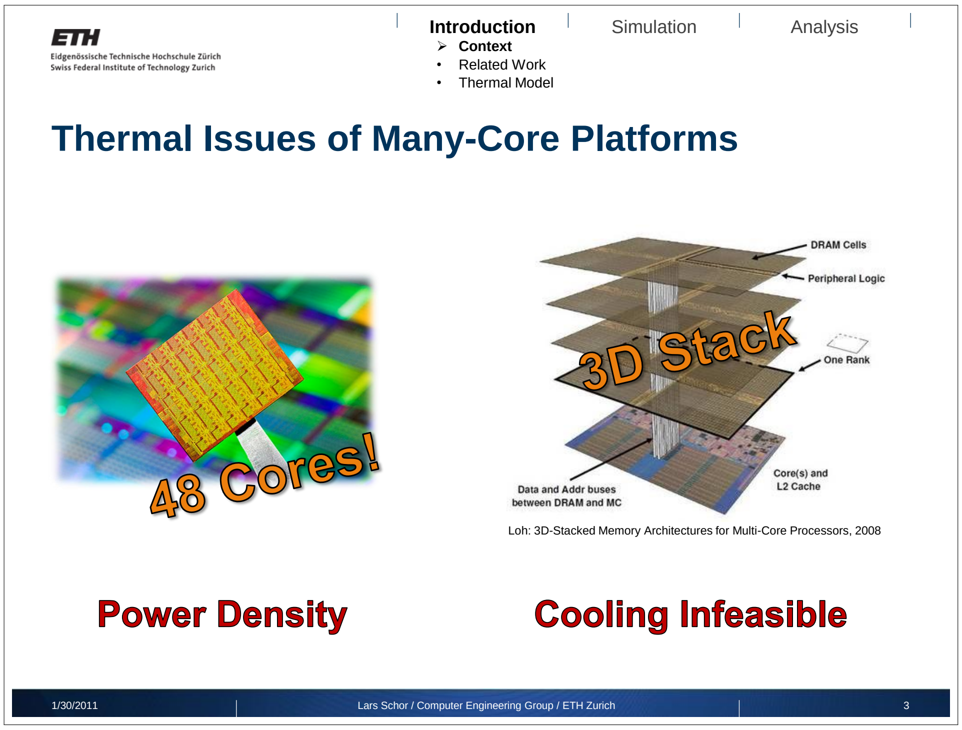


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

PRO3D

Thermal Simulation and Analysis Methods for Many-Core Platforms

Student:	Lars Schor
Advisors:	Dr. Iuliana Bacivarov, Dr. Hoeseok Yang
Professor:	Prof. Dr. Lothar Thiele



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Introduction | Simulation | Analysis

- **Context**
- Related Work
- Thermal Model

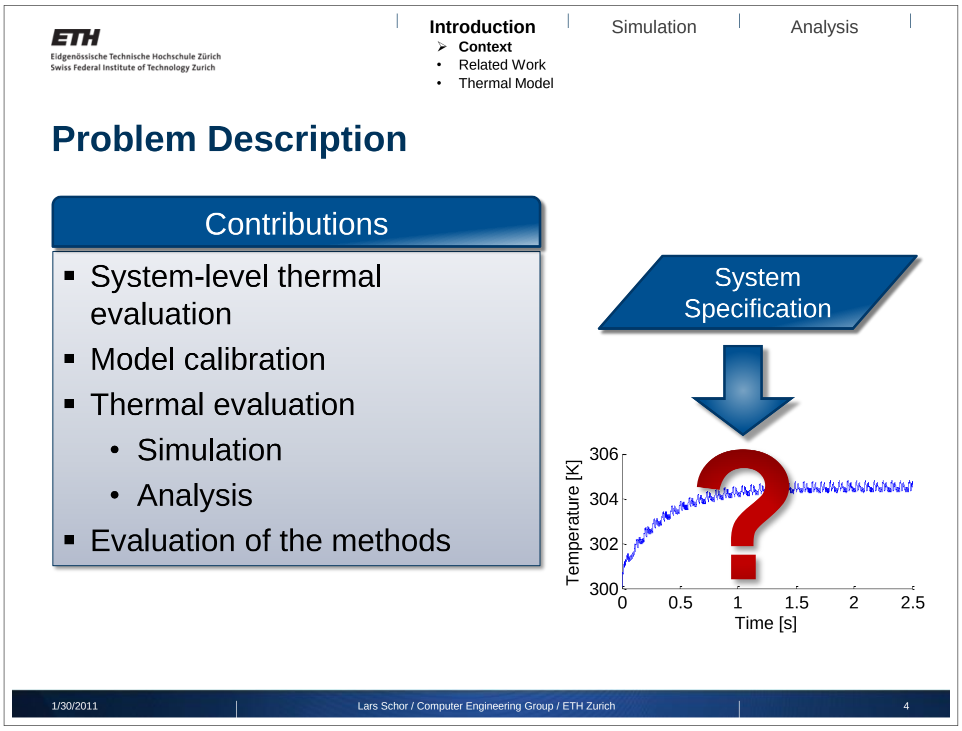
Thermal Issues of Many-Core Platforms

48 Cores!
DRAM Cells
Peripheral Logic
3D Stack
One Rank
Data and Addr buses between DRAM and MC
Core(s) and L2 Cache

Loh: 3D-Stacked Memory Architectures for Multi-Core Processors, 2008

Power Density **Cooling Infeasible**

1/30/2011 | Lars Schor / Computer Engineering Group / ETH Zurich | 3



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Introduction
- ➢ **Context**
- Related Work
- Thermal Model

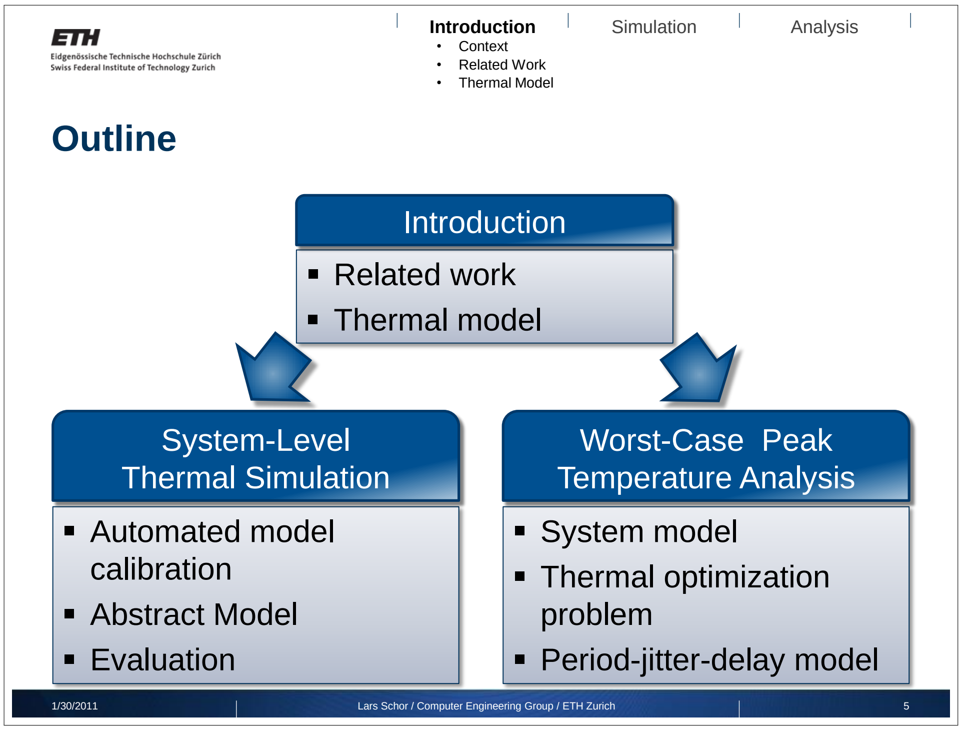
Simulation | Analysis

Problem Description

Contributions
- System-level thermal evaluation
- Model calibration
- Thermal evaluation
  - Simulation
  - Analysis
- Evaluation of the methods

System Specification
?
306
304
302
300
Temperature [K]
0
0.5
1
1.5
2
2.5
Time [s]

1/30/2011 | Lars Schor / Computer Engineering Group / ETH Zurich | 4



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Introduction
- Context
- Related Work
- Thermal Model

Simulation | Analysis

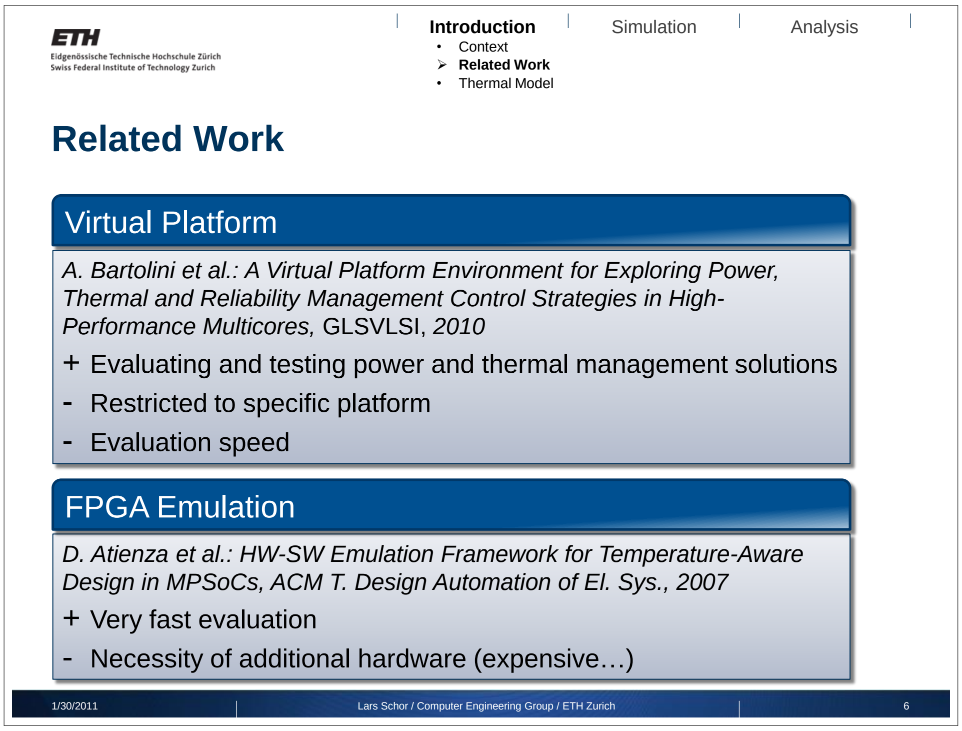
Outline

Introduction
- Related work
- Thermal model

System-Level Thermal Simulation
- Automated model calibration
- Abstract Model
- Evaluation

Worst-Case Peak Temperature Analysis
- System model
- Thermal optimization problem
- Period-jitter-delay model

1/30/2011 | Lars Schor / Computer Engineering Group / ETH Zurich | 5



ETH Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Introduction | Simulation | Analysis
- Context
- **Related Work**
- Thermal Model

Related Work

Virtual Platform

A. Bartolini et al.: A Virtual Platform Environment for Exploring Power, Thermal and Reliability Management Control Strategies in High-Performance Multicores, GLSVLSI, *2010*

+ Evaluating and testing power and thermal management solutions

- Restricted to specific platform

- Evaluation speed

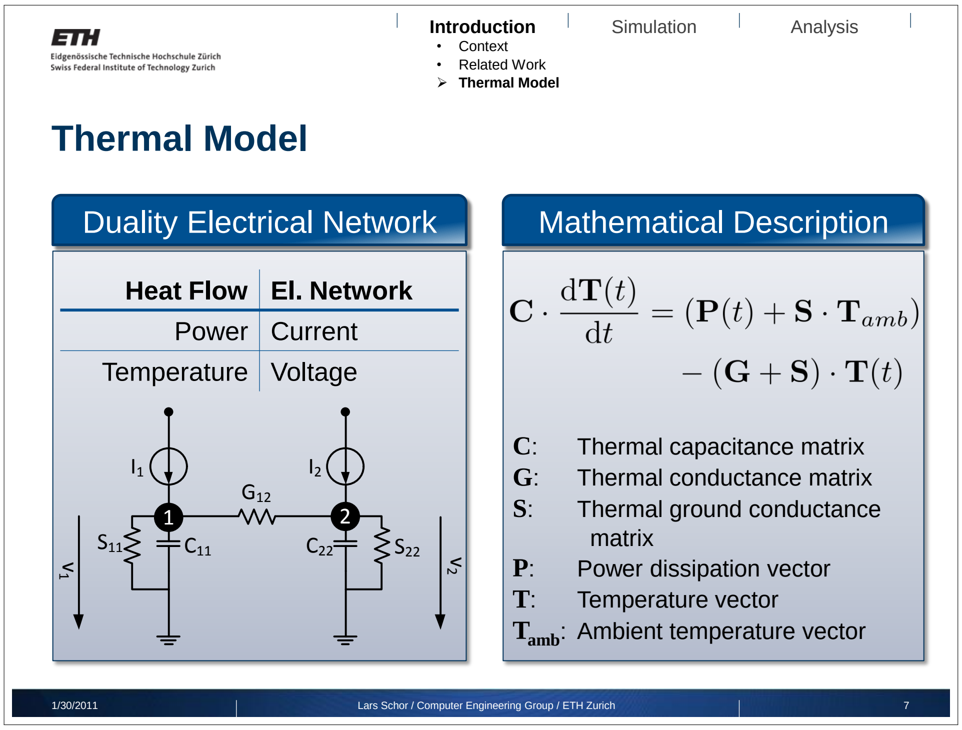
FPGA Emulation

D. Atienza et al.: HW-SW Emulation Framework for Temperature-Aware Design in MPSoCs, ACM T. Design Automation of El. Sys., 2007

+ Very fast evaluation

- Necessity of additional hardware (expensive…)

1/30/2011 | Lars Schor / Computer Engineering Group / ETH Zurich | 6



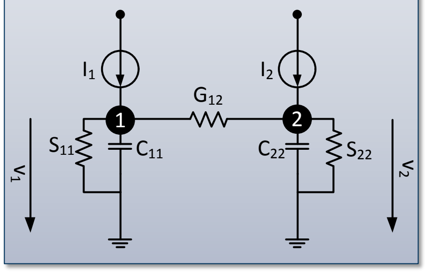
ETH Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Introduction | Simulation | Analysis
- Context
- Related Work
- **Thermal Model**

Thermal Model

Duality Electrical Network

Heat Flow	El. Network
Power	Current
Temperature	Voltage

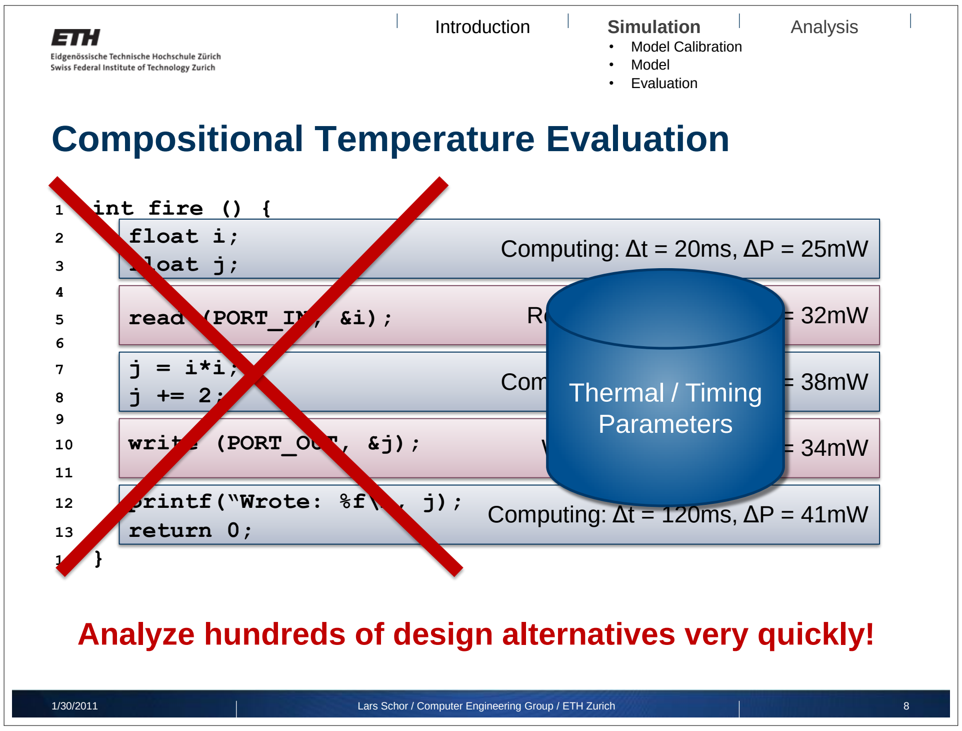


Mathematical Description

$$\mathbf{C} \cdot \frac{d\mathbf{T}(t)}{dt} = (\mathbf{P}(t) + \mathbf{S} \cdot \mathbf{T}_{amb}) - (\mathbf{G} + \mathbf{S}) \cdot \mathbf{T}(t)$$

$\mathbf{C}$:	Thermal capacitance matrix
$\mathbf{G}$:	Thermal conductance matrix
$\mathbf{S}$:	Thermal ground conductance matrix
$\mathbf{P}$:	Power dissipation vector
$\mathbf{T}$:	Temperature vector
$\mathbf{T}_{\mathbf{amb}}$:	Ambient temperature vector

1/30/2011 | Lars Schor / Computer Engineering Group / ETH Zurich | 7



ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Introduction | **Simulation** | Analysis
- Model Calibration
- Model
- Evaluation

Compositional Temperature Evaluation

```
int fire () {
  float i;
  float j;

  read (PORT_IN, &i);

  j = i*i;
  j += 2;

  write (PORT_OUT, &j);

  printf("Wrote: %f\n", j);
  return 0;
}
```

Computing: Δt = 20ms, ΔP = 25mW

R… = 32mW

Com… = 38mW

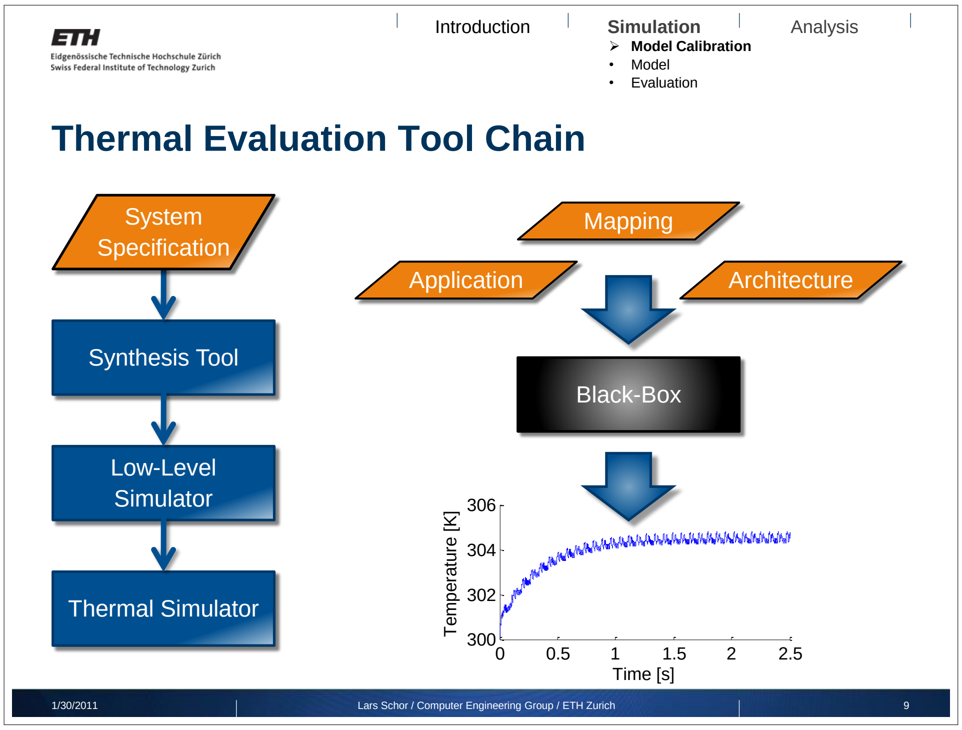
W… = 34mW

Computing: Δt = 120ms, ΔP = 41mW

Thermal / Timing Parameters

Analyze hundreds of design alternatives very quickly!

1/30/2011 | Lars Schor / Computer Engineering Group / ETH Zurich | 8



ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Introduction | **Simulation** | Analysis
- **Model Calibration**
- Model
- Evaluation

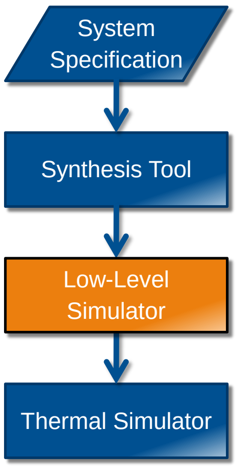
Thermal Evaluation Tool Chain

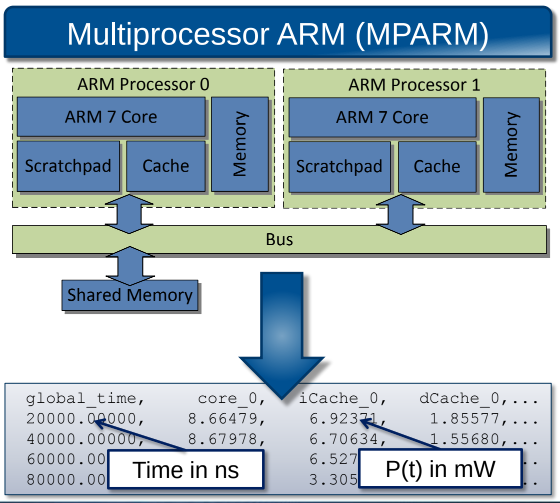
1/30/2011 | Lars Schor / Computer Engineering Group / ETH Zurich | 9

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Introduction | **Simulation** | Analysis
- **Model Calibration**
- Model
- Evaluation

Tool Chain: Simulation



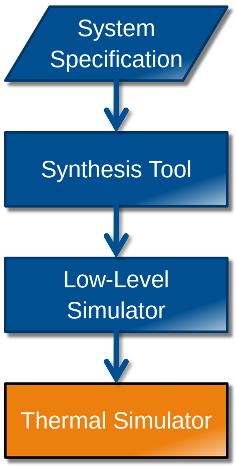


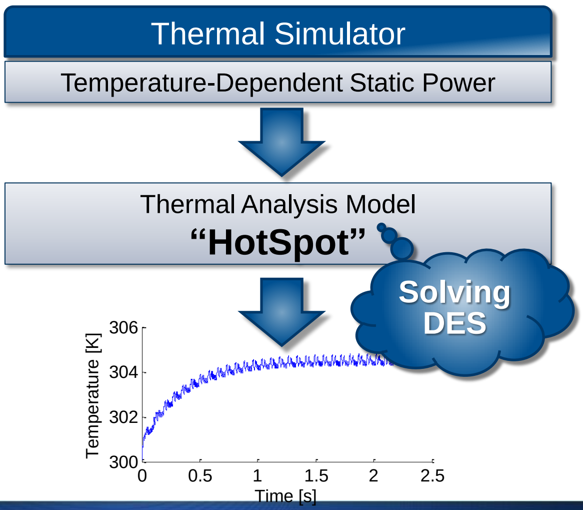
```
global_time,     core_0,    iCache_0,    dCache_0,...
20000.00000,   8.66479,     6.92371,     1.85577,...
40000.00000,   8.67978,     6.70634,     1.55680,...
60000.00                    6.527                 ..
80000.00                    3.305                 ..
```

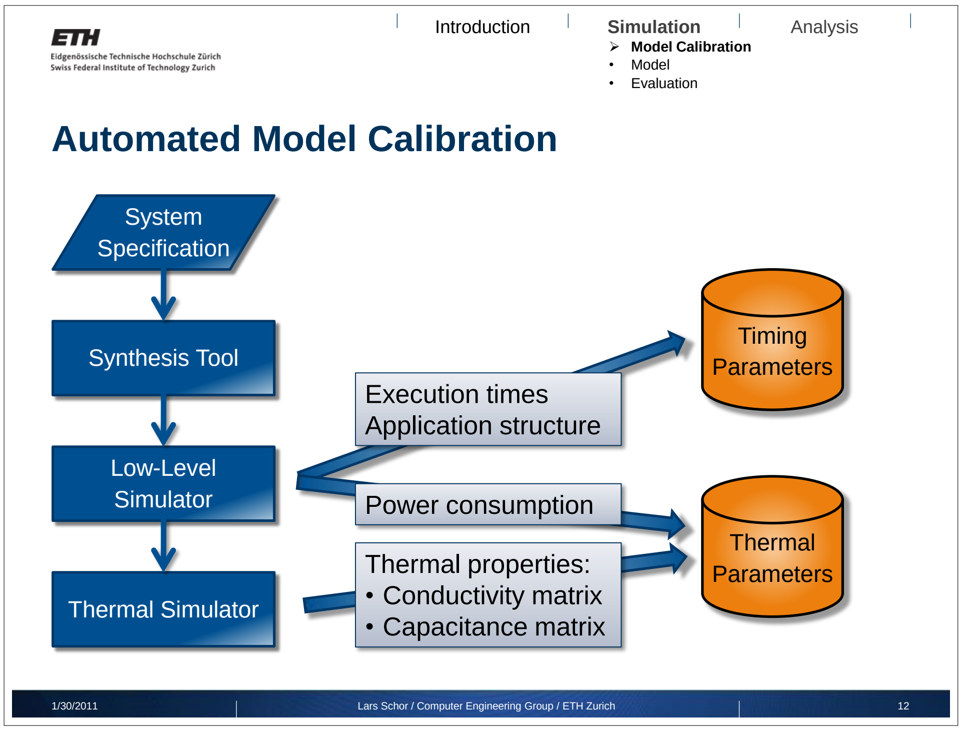
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Introduction | **Simulation** | Analysis
- **Model Calibration**
- Model
- Evaluation

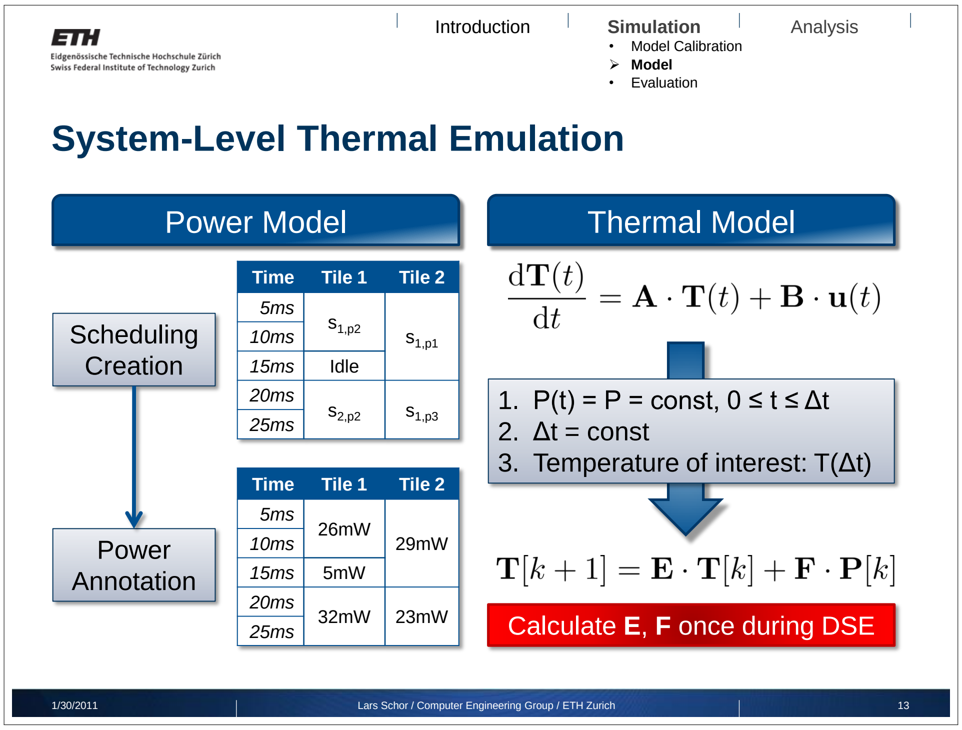
Tool Chain: Thermal Analysis







Automated Model Calibration



System-Level Thermal Emulation

Power Model

Scheduling Creation
Power Annotation

Time	Tile 1	Tile 2
5ms	$s_{1,p2}$	$s_{1,p1}$
10ms		
15ms	Idle	
20ms	$s_{2,p2}$	$s_{1,p3}$
25ms		

Time	Tile 1	Tile 2
5ms	26mW	29mW
10ms		
15ms	5mW	
20ms	32mW	23mW
25ms		

Thermal Model

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

1. P(t) = P = const, $0 \leq t \leq \Delta t$
2. Δt = const
3. Temperature of interest: $T(\Delta t)$

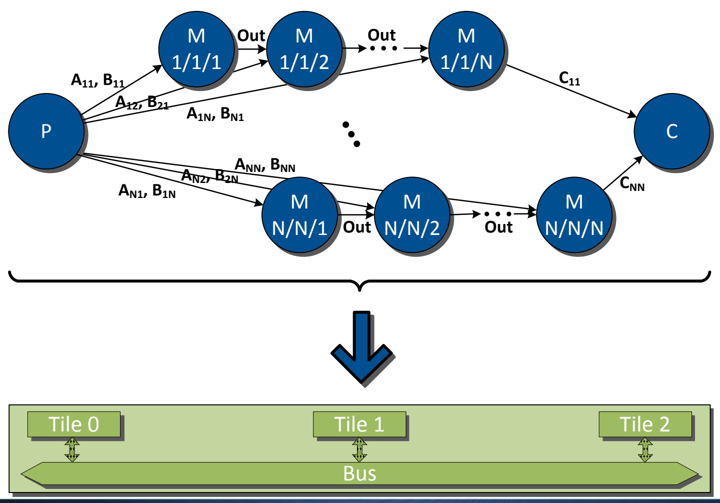
$$\mathbf{T}[k+1] = \mathbf{E} \cdot \mathbf{T}[k] + \mathbf{F} \cdot \mathbf{P}[k]$$

Calculate **E**, **F** once during DSE

Introduction | **Simulation** | Analysis

- Model Calibration
- Model
- ➢ **Evaluation**

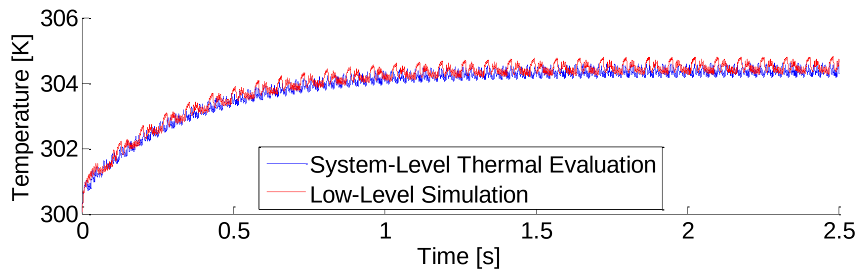
Evaluation: Matrix Multiplication

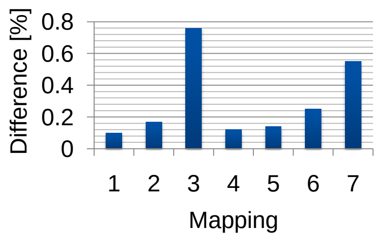


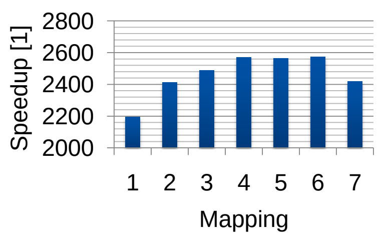
Introduction | **Simulation** | Analysis

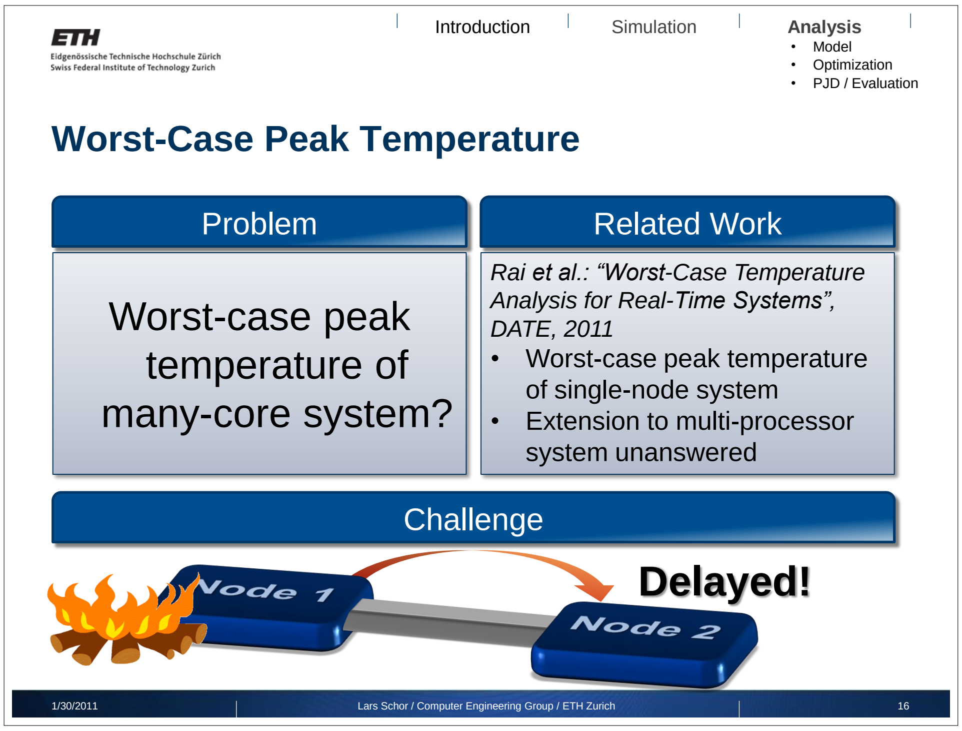
- Model Calibration
- Model
- ➢ **Evaluation**

Evaluation: Matrix Multiplication









ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Introduction | Simulation | **Analysis**
- Model
- Optimization
- PJD / Evaluation

Worst-Case Peak Temperature

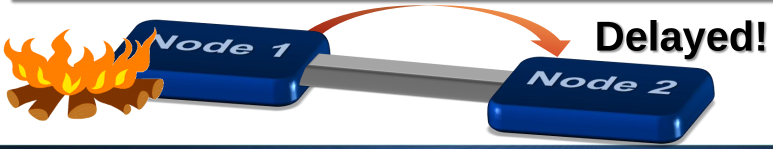
Problem

Worst-case peak temperature of many-core system?

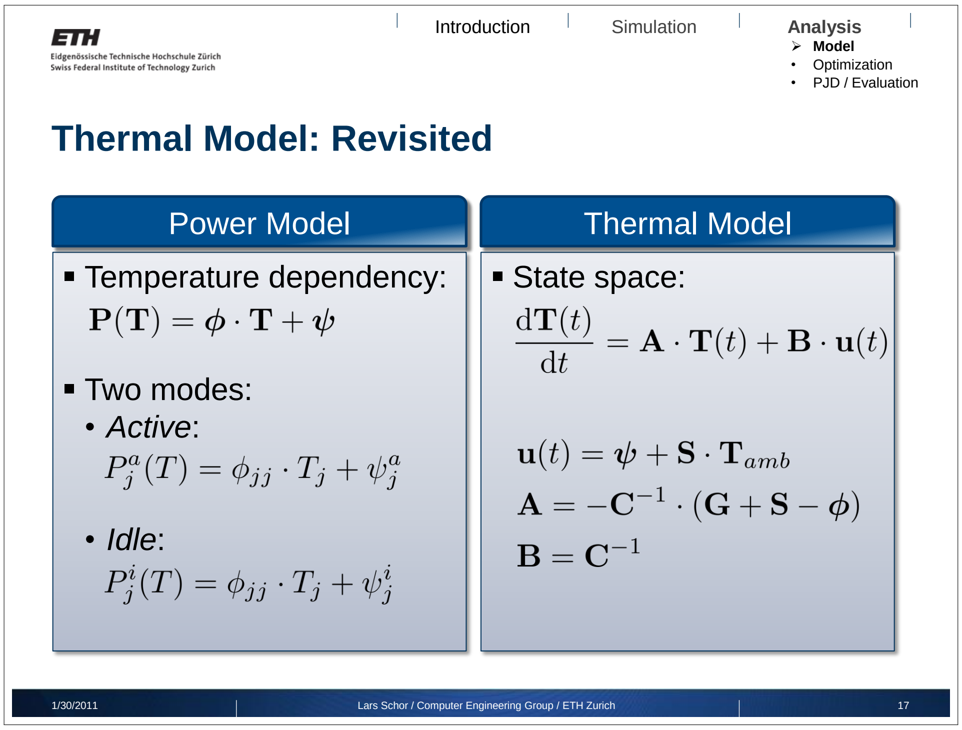
Related Work

Rai et al.: "Worst-Case Temperature Analysis for Real-Time Systems", DATE, 2011
- Worst-case peak temperature of single-node system
- Extension to multi-processor system unanswered

Challenge



1/30/2011 | Lars Schor / Computer Engineering Group / ETH Zurich | 16



ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Introduction | Simulation | **Analysis**
- **Model**
- Optimization
- PJD / Evaluation

Thermal Model: Revisited

Power Model

- Temperature dependency:

$$\mathbf{P}(\mathbf{T}) = \boldsymbol{\phi} \cdot \mathbf{T} + \boldsymbol{\psi}$$

- Two modes:
  - *Active*:

    $$P_j^a(T) = \phi_{jj} \cdot T_j + \psi_j^a$$

  - *Idle*:

    $$P_j^i(T) = \phi_{jj} \cdot T_j + \psi_j^i$$

Thermal Model

- State space:

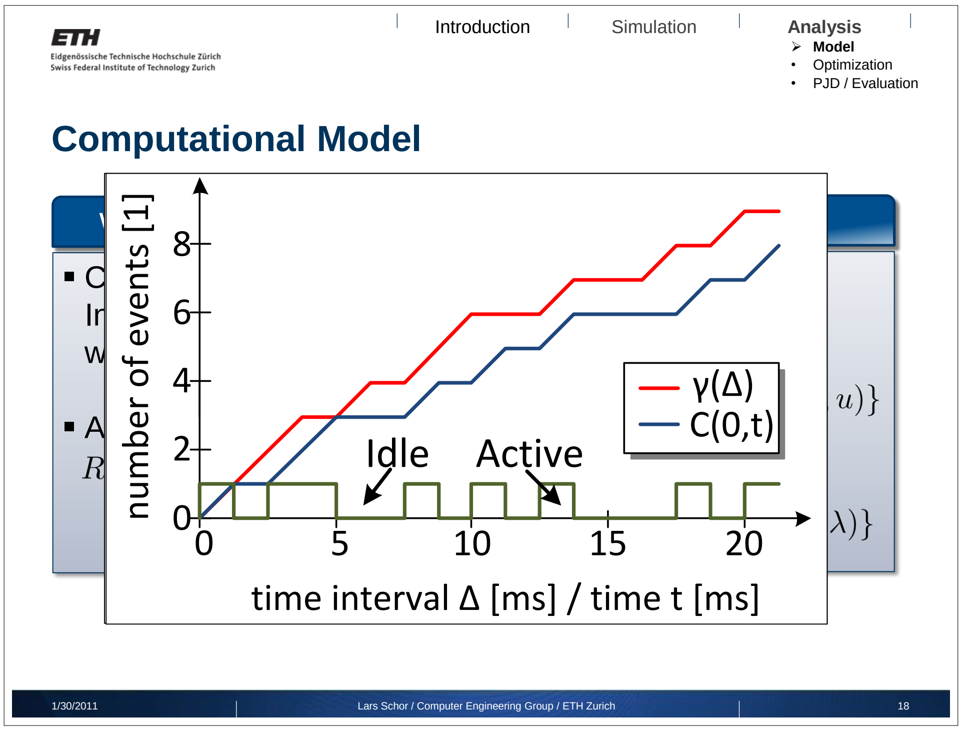
$$\frac{\mathrm{d}\mathbf{T}(t)}{\mathrm{d}t} = \mathbf{A} \cdot \mathbf{T}(t) + \mathbf{B} \cdot \mathbf{u}(t)$$

$$\mathbf{u}(t) = \boldsymbol{\psi} + \mathbf{S} \cdot \mathbf{T}_{amb}$$

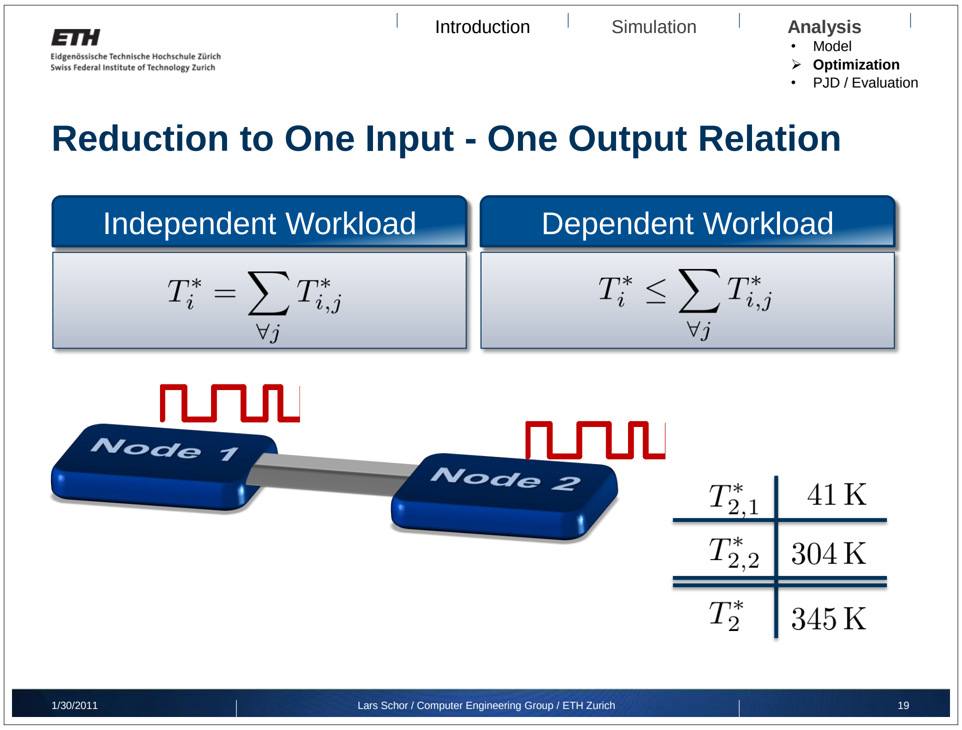
$$\mathbf{A} = -\mathbf{C}^{-1} \cdot (\mathbf{G} + \mathbf{S} - \boldsymbol{\phi})$$

$$\mathbf{B} = \mathbf{C}^{-1}$$

1/30/2011 | Lars Schor / Computer Engineering Group / ETH Zurich | 17



Computational Model

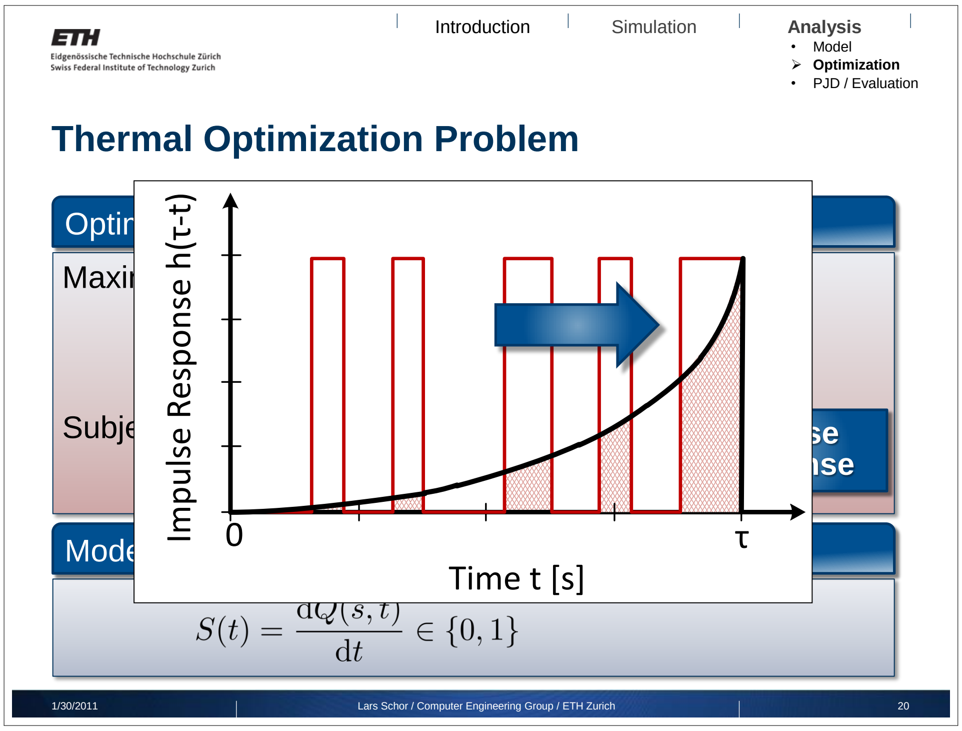


Reduction to One Input - One Output Relation

Independent Workload	Dependent Workload
$T_i^* = \sum_{\forall j} T_{i,j}^*$	$T_i^* \leq \sum_{\forall j} T_{i,j}^*$

Node 1
Node 2

$T_{2,1}^*$	41 K
$T_{2,2}^*$	304 K
T_2^*	345 K



ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

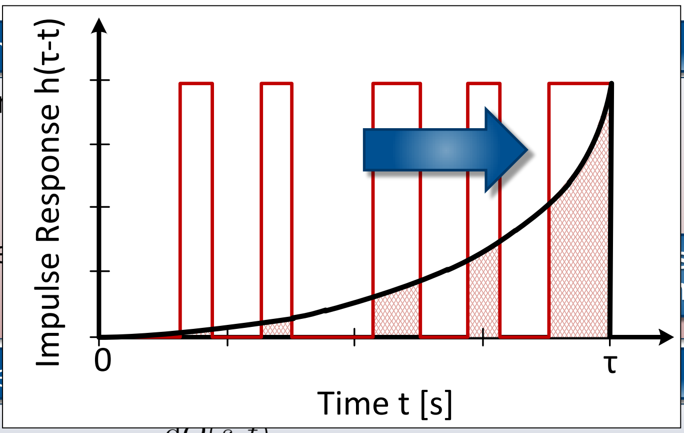
Introduction | Simulation | Analysis
- Model
- Optimization
- PJD / Evaluation

Thermal Optimization Problem

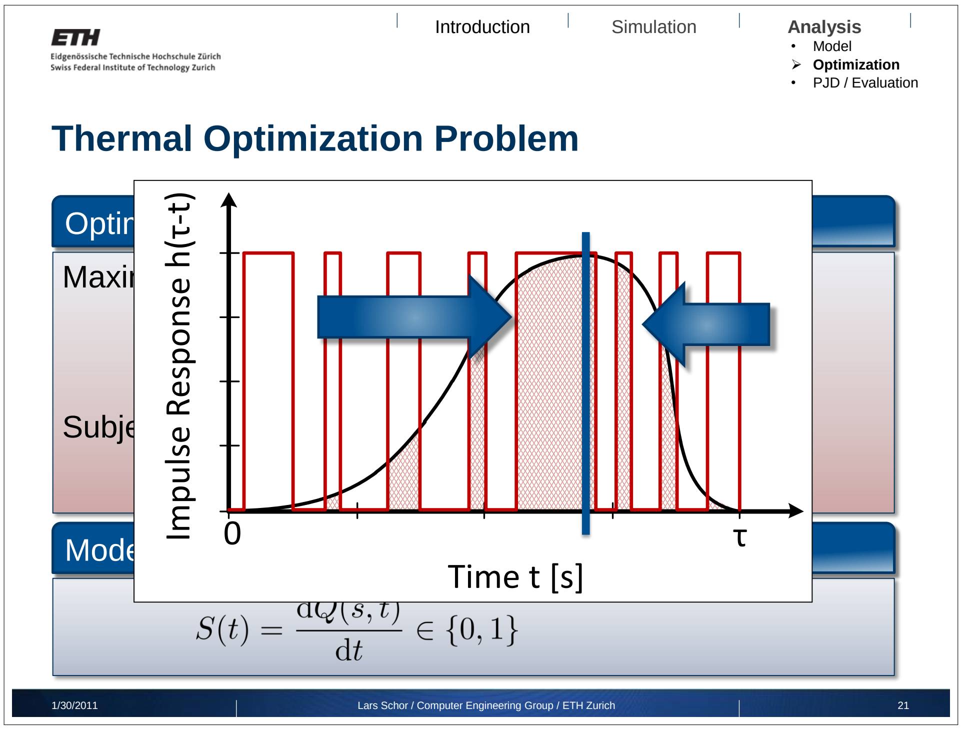
Optir
Maxi
Subje
Mode

$$S(t) = \frac{\mathrm{d}Q(s,t)}{\mathrm{d}t} \in \{0,1\}$$

se
nse



1/30/2011 | Lars Schor / Computer Engineering Group / ETH Zurich | 20



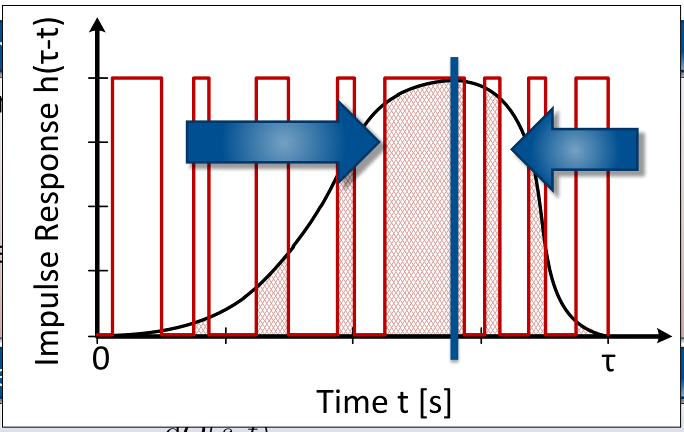
ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Introduction | Simulation | Analysis
- Model
- Optimization
- PJD / Evaluation

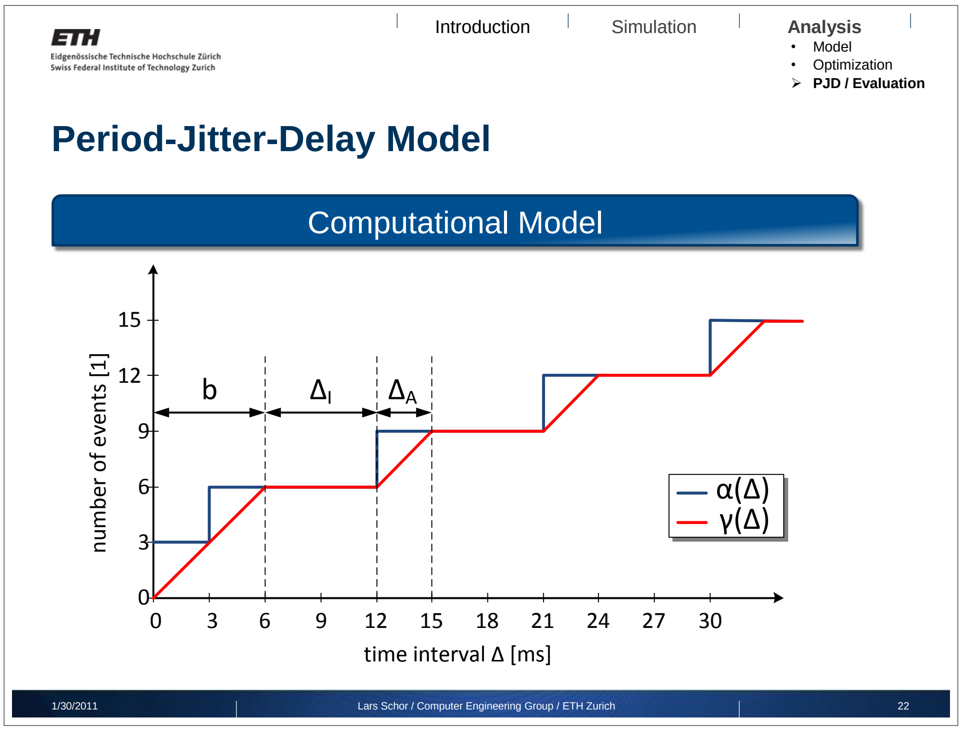
Thermal Optimization Problem

Optir
Maxi
Subje
Mode

$$S(t) = \frac{\mathrm{d}Q(s,t)}{\mathrm{d}t} \in \{0,1\}$$



1/30/2011 | Lars Schor / Computer Engineering Group / ETH Zurich | 21

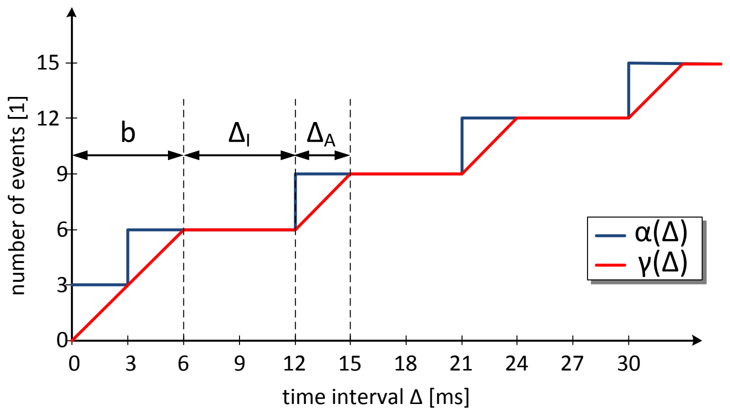


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

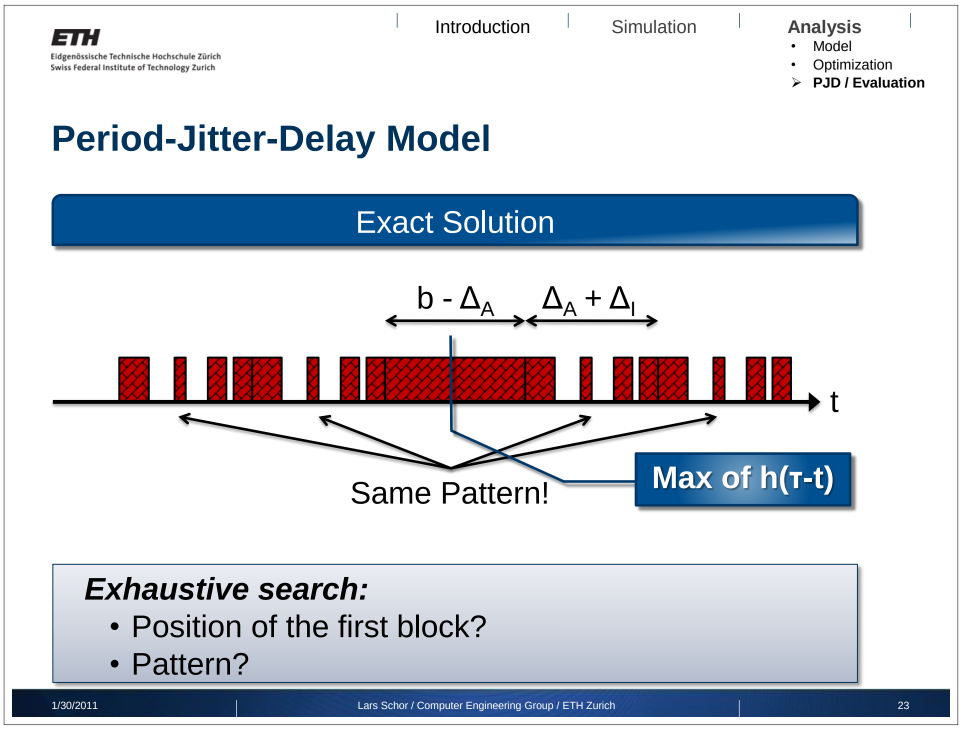
Introduction | Simulation | Analysis
- Model
- Optimization
- ➢ **PJD / Evaluation**

Period-Jitter-Delay Model

Computational Model



1/30/2011 | Lars Schor / Computer Engineering Group / ETH Zurich | 22

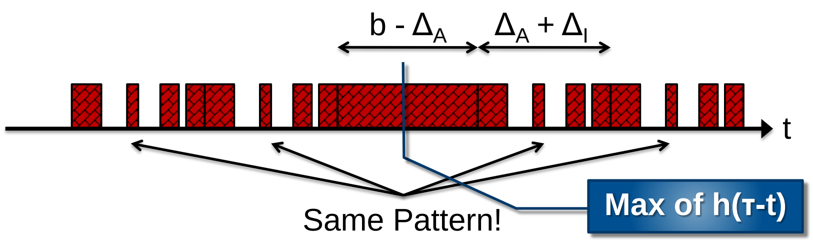


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Introduction | Simulation | Analysis
- Model
- Optimization
- ➢ **PJD / Evaluation**

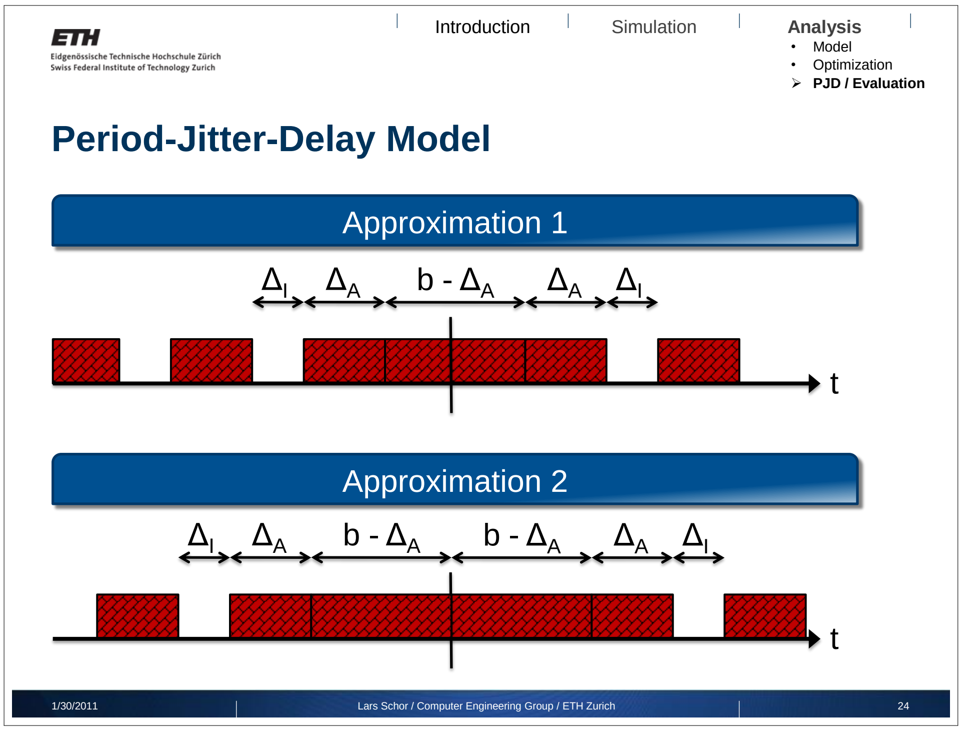
Period-Jitter-Delay Model

Exact Solution



Exhaustive search:
- Position of the first block?
- Pattern?

1/30/2011 | Lars Schor / Computer Engineering Group / ETH Zurich | 23

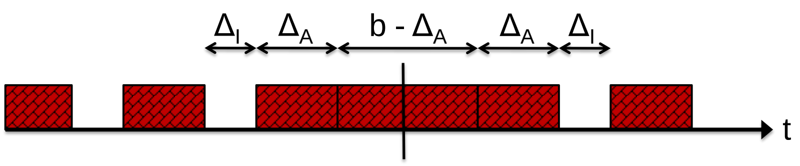


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

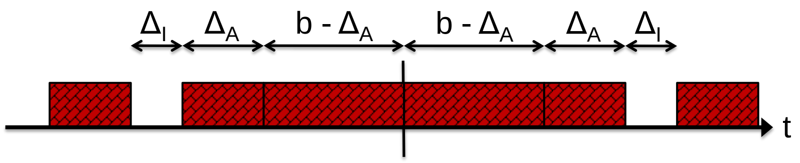
Introduction | Simulation | **Analysis**
- Model
- Optimization
- ➢ **PJD / Evaluation**

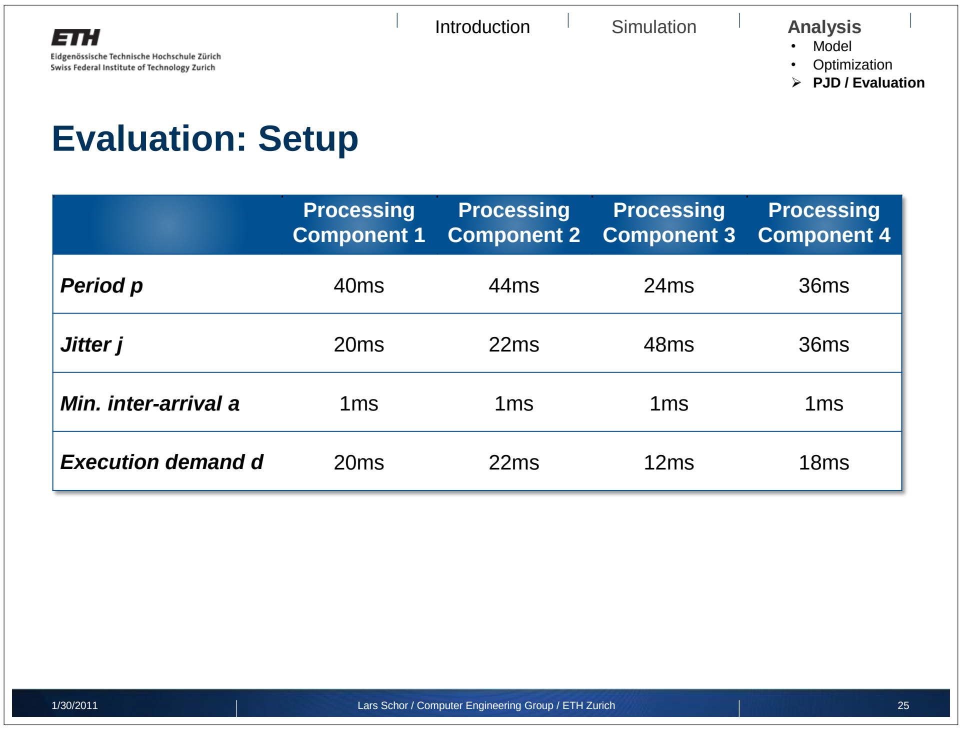
Period-Jitter-Delay Model

Approximation 1



Approximation 2





ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Introduction | Simulation | **Analysis**
- Model
- Optimization
- ➢ **PJD / Evaluation**

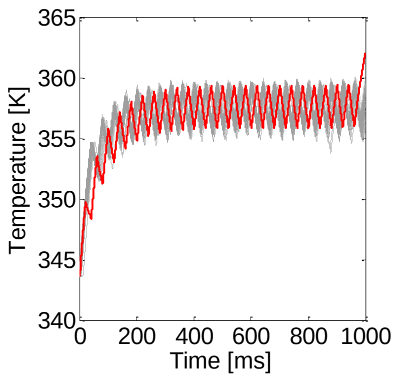
Evaluation: Setup

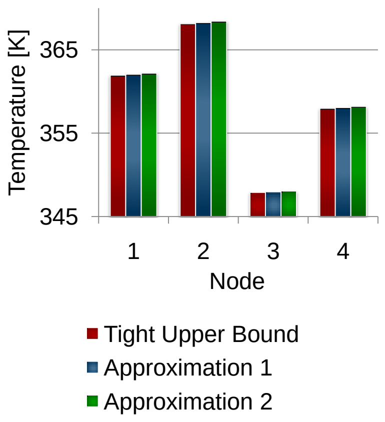
	Processing Component 1	Processing Component 2	Processing Component 3	Processing Component 4
Period p	40ms	44ms	24ms	36ms
Jitter j	20ms	22ms	48ms	36ms
Min. inter-arrival a	1ms	1ms	1ms	1ms
Execution demand d	20ms	22ms	12ms	18ms

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Evaluation: Result

Random Trace | **Approximation**

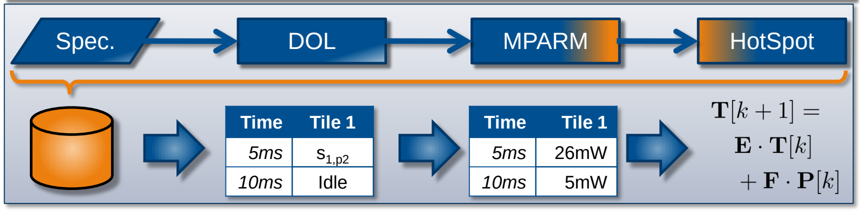




ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Conclusion

System-Level Thermal Simulation



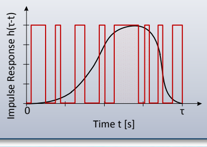
Time	Tile 1
5ms	$s_{1,p2}$
10ms	Idle

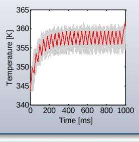
Time	Tile 1
5ms	26mW
10ms	5mW

$$\mathbf{T}[k+1] = \mathbf{E}\cdot\mathbf{T}[k] + \mathbf{F}\cdot\mathbf{P}[k]$$

Worst-Case Peak Temperature Analysis

$$T_i^* = \sum_{\forall j} T_{i,j}^*$$





C
List of Symbols

$\mathcal{T}$	Architecture specification.
$\mathcal{A}$	Application specification.
$b = (b_1; \ldots; b_{\lvert V \rvert})$	Binding.
ψ	Calibration data.
Q	Set of channels.
$q \in Q$	Channel.
c	Component.
I	Set of instructions.
$i \in I$	Instruction.
(b, s)	Mapping specification.
P	Power.
$\mathcal{P}\{v\}$	Priority of v.
V	Set of processes.
$v \in V$	Process.
s	Scheduling.
S_p	Set of segments of process p.
$s_{i,p} \in S_p$	Segment i of process p.
sc	Shared component.
$\mathcal{S}$	System specification.
T	Temperature.

Γ	Set of tiles.
$\gamma \in \Gamma$	Tile.
t	Time.
Θ	Set of timetables.
$\theta \in \Theta$	Timetable.
$\mathfrak{T}(\cdot)$	Type of.

D
Acronyms

ACET	Average Case Execution Time.
AMBA	Advanced Microcontroller Bus Architecture.
BCET	Best Case Execution Time.
DMA	Direct Memory Access.
DOL	Distributed Operation Layer.
DRAM	Dynamic Random Access Memory.
DVFS	Dynamic Voltage and Frequency Scaling.
FFT	Fast Fourier Transform.
FIFO	First-In First-Out.
FPGA	Field Programmable Gate Array.
IC	Integrated Circuit.
IDCT	Inverse Discrete Cosine Transformation.
IIR	Infinite Impulse Response.
IQ	Inverse Quantization.
ISS	Instruction-Set-Simulator.
KPN	Kahn Process Network.
MC	Motion Compensation.
MJPEG	Motion JPEG.

MPA	Modular Performance Analysis.
MPARM	Multiprocessor ARM.
MPSoC	Multiprocessor System-on-Chip.
MTJ	Matrix-Toolkits-Java.
NoC	Network-on-a-Chip.
OS	Operating System.
RTEMS	Real-Time Executive for Multiprocessor Systems.
SCC	Single-chip Cloud Computer.
SDF	Synchronous Dataflow.
SIMO	Single-Input and Multiple-Output.
SLTE	System Level Thermal Emulator.
SoC	System On Chip.
SWARM	Software ARM.
SymTA/S	Symbolic Timing Analysis for Systems.
VLD	Variable Length Decoding.
WCET	Worst Case Execution Time.

Bibliography

[1] C. R. Johns and D. A. Brokenshire, “Introduction to the Cell Broadband Engine Architecture,” *IBM J. of Research and Dev.*, vol. 51, no. 5, pp. 503–519, Sep. 2007.

[2] “Single-Chip Cloud Computer,” Dec. 2010. [Online]. Available: http://techresearch.intel.com/ProjectDetails.aspx?Id=1

[3] I. Bacivarov, W. Haid, K. Huang, and L. Thiele, “Methods and Tools for Mapping Process Networks onto Multi-Processor Systems-On-Chip,” in *Handbook of Signal Processing Systems*, S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, Eds. Springer, Oct. 2010, pp. 1007–1040.

[4] L. Thiele, I. Bacivarov, W. Haid, and K. Huang, “Mapping Applications to Tiled Multiprocessor Embedded Systems,” in *Proc. Int’l Conf. on Application of Concurrency to System Design (ACSD)*, Jul. 2007, pp. 29–40.

[5] S. Chakraborty, S. Kunzli, and L. Thiele, “A General Framework for Analysing System Properties in Platform-Based Embedded System Designs,” in *Proc. Design, Automation and Test in Europe (DATE)*, 2003, pp. 190–195.

[6] C. Silvano, W. Fornaciari, S. C. Reghizzi, G. Agosta, G. Palermo, V. Zaccaria, P. Bellasi, F. Castro, S. Corbetta, A. D. Biagio, E. Speziale, M. Tartara, D. Siorpaes, H. Hubert, B. Stabernack, J. Brandenburg, M. Palkovic, P. Raghavan, C. Ykman-Couvreur, A. Bartzas, S. Xydis, D. Soudris, T. Kempf, G. Ascheid, R. Leupers, H. Meyr, J. Ansari, P. Mahonen, and B. Vanthournout, “2PARMA: Parallel Paradigms and Run-Time Management Techniques for Many-Core Architectures,” in *Proc. IEEE Annual Symposium on VLSI (ISVLSI)*, 2010, pp. 494–499.

[7] T. Sporer, M. Beckinger, A. Franck, I. Bacivarov, W. Haid, K. Huang, L. Thiele, P. S. Paolucci, P. Bazzana, P. Vicini, J. Ceng, S. Kraemer, and R. Leupers, “SHAPES — A Scalable Parallel HW/SW Architecture Applied to Wave Field Synthesis,” in *Proc. Int’l Audio Engineering Society (AES) Conf.*, Sep. 2007, pp. 175–187.

[8] B. Kienhuis, E. Deprettere, K. Vissers, and P. Van Der Wolf, “An Approach for Quantitative Analysis of Application-Specific Dataflow

Architectures," in *Proc. IEEE Int'l Conf. on Application-Specific Systems, Architectures and Processors*, 1997, pp. 338–349.

[9] G. Kahn, "The Semantics of a Simple Language for Parallel Programming," in *Proc. of the IFIP Congress*, vol. 74, 1974, pp. 471–475.

[10] K. Huang, W. Haid, I. Bacivarov, and L. Thiele, "Coupling MPARM with DOL," ETH Zürich, TIK Report 314, Sep. 2009.

[11] S. Künzli, "Efficient Design Space Exploration for Embedded Systems," Ph.D. dissertation, ETH Zurich, 2006.

[12] H. Yang and S. Ha, "Pipelined Data Parallel Task Mapping/Scheduling Technique for MPSoC," in *Proc. Design, Automation and Test in Europe (DATE)*, 2009, pp. 69–74.

[13] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, and M. Olivieri, "MPARM: Exploring the Multi-Processor SoC Design Space with SystemC," *J. VLSI Signal. Proces.*, vol. 41, no. 2, pp. 169–182, Sep. 2005.

[14] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. Stan, "HotSpot: A Compact Thermal Modeling Methodology for Early-Stage VLSI Design," *IEEE T. VLSI Sys.*, vol. 14, no. 5, pp. 501–513, May 2006.

[15] D. Rai, H. Yang, I. Bacivarov, J.-J. Chen, and L. Thiele, "Worst-Case Temperature Analysis for Real-Time Systems," in *Proc. Design, Automation and Test in Europe (DATE)*, Mar. 2011.

[16] S. Perathoner, E. Wandeler, L. Thiele, A. Hamann, S. Schliecker, R. Henia, R. Racu, R. Ernst, and M. González Harbour, "Influence of Different Abstractions on the Performance Analysis of Distributed Hard Real-Time Systems," *Design Automation for Embedded Systems*, vol. 13, pp. 27–49, 2009.

[17] M. Sridhar, A. Raj, A. Vincenzi, M. Ruggiero, T. Brunschwiler, and D. Atienza Alonso, "3D-ICE: Fast Compact Transient Thermal Modeling For 3D-ICs with Inter-tier Liquid Cooling," in *Proc. Int'l Conf. on Computer-Aided Design (ICCAD)*, 2010, pp. 463–470.

[18] K. Puttaswamy and G. H. Loh, "Thermal Analysis of a 3D Die-Stacked High-Performance Microprocessor," in *Proc. Great Lakes Symposium on VLSI (GLSVLSI)*, 2006, pp. 19–24.

[19] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. P. Boyd, and G. De Micheli, "Temperature-Aware Processor Frequency Assignment for MPSoCs Using Convex Optimization," in *Proc. IEEE/ACM Int'l Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2007, pp. 111–116.

[20] J. Donald and M. Martonosi, "Techniques for Multicore Thermal Management: Classification and New Exploration," in *Proc. Annual Int'l Symposium on Computer Architecture (ISCA)*, 2006, pp. 78–88.

[21] A. K. Coskun, T. S. Rosing, and K. Whisnant, "Temperature Aware Task Scheduling in MPSoCs," in *Proc. Design, Automation and Test in Europe (DATE)*, 2007, pp. 1659–1664.

[22] A. K. Coskun, J. L. Ayala, D. Atienza, T. Simunic, and Y. Leblebici, "Dynamic Thermal Management in 3D Multicore Architectures," in *Proc. Design, Automation and Test in Europe (DATE)*, 2009, pp. 1410–1415.

[23] J. Yang, X. Zhou, M. Chrobak, Y. Zhang, and L. Jin, "Dynamic Thermal Management Through Task Scheduling," in *Proc. IEEE Int'l Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2008, pp. 191–201.

[24] M. Sabry, J. Ayala, and D. Atienza, "Thermal-Aware Compilation for System-on-Chip Processing Architectures," in *Proc. Great Lakes Symposium on VLSI (GLSVLSI)*, 2010, pp. 221–226.

[25] T. Chantem, R. P. Dick, and X. S. Hu, "Temperature-Aware Scheduling and Assignment for Hard Real-Time Applications on MPSoCs," in *Proc. Design, Automation and Test in Europe (DATE)*, 2008, pp. 288–293.

[26] Y. Xie and W.-l. Hung, "Temperature-Aware Task Allocation and Scheduling for Embedded Multiprocessor Systems-on-Chip (MPSoC) Design," *J. VLSI Signal. Proces.*, vol. 45, no. 3, pp. 177–189, Dec. 2006.

[27] W.-L. Hung, Y. Xie, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "Thermal-Aware Task Allocation and Scheduling for Embedded Systems," in *Proc. Design, Automation and Test in Europe (DATE)*, vol. 2, 2005, pp. 898–899.

[28] A. Pimentel, "The Artemis Workbench for System-Level Performance Evaluation of Embedded Systems," *Int'l J. Embedded Systems*, vol. 3, no. 3, pp. 181–196, 2008.

[29] W. Haid, M. Keller, K. Huang, I. Bacivarov, and L. Thiele, "Generation and Calibration of Compositional Performance Analysis Models for Multi-Processor Systems," in *Proc. Int'l Conf. on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, Jul. 2009, pp. 92–99.

[30] T. Grötker, S. Liao, G. Martin, and S. Swan, *System Design with SystemC*. Springer, 2002.

[31] M. Loghi, M. Poncino, and L. Benini, "Cycle-Accurate Power Analysis for Multiprocessor Systems-on-a-Chip," in *Proc. Great Lakes Symposium on VLSI (GLSVLSI)*, 2004, pp. 406–410.

[32] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," in *Proc. Annual Int'l Symposium on Computer Architecture (ISCA)*, vol. 28, no. 2, 2000, pp. 83–94.

[33] N. Eisley, V. Soteriou, and L. Peh, "High-Level Power Analysis for Multi-Core Chips," in *Proc. Int'l Conf. on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, 2006, pp. 389–400.

[34] Y. Nakamura, K. Hosokawa, I. Kuroda, K. Yoshikawa, and T. Yoshimura, "A Fast Hardware/Software Co-Verification Method for System-on-a-Chip by Using a C/C++ Simulator and FPGA Emulator with Shared Register Communication," in *Proc. Annual Design Automation Conf. (DAC)*, 2004, pp. 299–304.

[35] D. Atienza, P. Del Valle, G. Paci, F. Poletti, L. Benini, G. Micheli, J. Mendias, and R. Hermida, "HW-SW Emulation Framework for Temperature-Aware Design in MPSoCs," *ACM T. Design Automation of Electronic Systems*, vol. 12, no. 3, pp. 1–26, 2007.

[36] C.-Y. Yang, J.-J. Chen, L. Thiele, and T.-W. Kuo, "Energy-Efficient Real-Time Task Scheduling with Temperature-Dependent Leakage," in *Proc. Design, Automation and Test in Europe (DATE)*, Mar. 2010, pp. 9–14.

[37] W. Huang, K. Skadron, R. Gurumurthi, R. J. Ribando, and M. R. Stan, "Differentiating the Roles of IR Measurement and Simulation for Power and Temperature-Aware Design," in *Proc. IEEE Int'l Symposium on Performance Analysis of Systems and Software (ISPASS)*, Apr. 2009.

[38] M. Girault and D. Petit, "Identification Methods In Nonlinear Heat Conduction. Part I: Model Reduction," *Int'l J. Heat and Mass Transfer*, vol. 48, no. 1, pp. 105–118, 2005.

[39] A. Augustin, T. Hauck, B. Maj, J. Czernohorsky, E. Rudnyi, and J. Korvink, "Model Reduction for Power Electronics Systems with Multiple Heat Sources," in *Proc. Int'l Workshop on Thermal investigations of ICs (THERMINIC)*, 2006, pp. 113–117.

[40] Y. Han, I. Koren, and C. Krishna, "Temptor: A Lightweight Runtime Temperature Monitoring Tool Using Performance Counters," in *Proc. Workshop on Temperature-Aware Computer Systems (TACS)*, 2006.

[41] A. Bartolini, M. Cacciari, A. Tilli, L. Benini, and M. Gries, "A Virtual Platform Environment for Exploring Power, Thermal and Reliability Management Control Strategies in High-Performance Multicores," in *Proc. Great Lakes Symposium on VLSI (GLSVLSI)*, 2010, pp. 311–316.

[42] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, and B. Werner, “Simics: A Full System Simulation Platform,” *Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[43] P. Garcia del Valle and D. Atienza, “Emulation-Based Transient Thermal Modeling of 2D/3D Systems-on-Chip with Active Cooling,” *Microelectronics J.*, vol. 41, no. 10, pp. 1–9, 2010.

[44] X. Zhu and S. Malik, “Using a Communication Architecture Specification in an Application-Driven Retargetable Prototyping Platform for Multiprocessing,” in *Proc. Design, Automation and Test in Europe (DATE)*, vol. 2, 2004, pp. 1244–1249.

[45] N. S. Kim, T. Austin, T. Mudge, and D. Grunwald, “Challenges For Architectural Level Power Modeling,” in *Power Aware Computing*, ser. Computer Science, R. Graybill and R. Melhem, Eds. Plenum Pub Corp, 2002, pp. 317–337.

[46] D. Gelernter and N. Carriero, “Coordination Languages and Their Significance,” *Commun. ACM*, vol. 35, pp. 97–107, Feb. 1992.

[47] M. Dales, “SWARM - Software ARM,” Feb. 2003. [Online]. Available: http://www.cl.cam.ac.uk/~mwd24/phd/swarm.html

[48] “RTEMS Operating System,” Jul. 2010. [Online]. Available: http://www.rtems.com/

[49] M. Caldari, A. Bona, V. Zaccaria, and R. Zafalon, “High-Level Power Characterization of the AMBA Bus Interconnect,” Synopsys User Group, Tech. Rep., 2004.

[50] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, “Temperature-Aware Microarchitecture: Modeling and Implementation,” *ACM T. Arch. and Code Opt.*, vol. 1, no. 1, pp. 94–125, 2004.

[51] A. Krum, “Thermal Management,” in *The CRC Handbook of Thermal Engineering*, F. Kreith, Ed. CRC Press, 2000, pp. 1–92.

[52] N. S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan, “Leakage Current: Moore’s Law Meets Static Power,” *Computer*, vol. 36, no. 12, pp. 68–75, 2003.

[53] F. Fallah and M. Pedram, “Standby and Active Leakage Current Control and Minimization in CMOS VLSI Circuits,” *IEICE T. Electron.*, no. 4, pp. 509–519, 2005.

[54] C. Hu, “BSIM3,” Mar. 2009. [Online]. Available: http://www-device.eecs.berkeley.edu/~bsim3/bsim_ent.html

[55] S. Heo, K. Barr, and K. Asanović, "Reducing Power Density Through Activity Migration," in *Proc. Int'l Symposium on Low Power Electronics and Design (ISLPED)*, 2003, pp. 217–222.

[56] Y. Liu, R. P. Dick, L. Shang, and H. Yang, "Accurate Temperature-Dependent Integrated Circuit Leakage Power Estimation is Easy," in *Proc. Design, Automation and Test in Europe (DATE)*, 2007, pp. 1526–1531.

[57] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan, "HotLeakage: A Temperature-Aware Model of Subthreshold and Gate Leakage for Architects," Univ. of Virginia, Dept. of Computer Science, Tech. Rep., Mar. 2003.

[58] K. Sankaranarayanan, "Thermal Modeling and Management of Microprocessors," Ph.D. dissertation, School of Engineering and Applied Science, University of Virginia, May 2009.

[59] *International Technology Roadmap for Semiconductors: Process Integration, Devices, and Structures*, SIAI, 2001. [Online]. Available: http://www.itrs.net/Links/2001ITRS/Home.htm

[60] Y. Cheng and C. Hu, *MOSFET Modeling and Bsim3 User's Guide.* Kluwer Academic Publishers, 1999.

[61] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System Level Performance Analysis - The SymTA/S Approach," *IEEE Proc. Comp. and Digital Techniques*, vol. 152, no. 2, pp. 148–166, 2005.

[62] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse, "System Architecture Evaluation Using Modular Performance Analysis: A Case Study," *Int'l J. Software Tools for Technology Transfer*, vol. 8, no. 6, pp. 649–667, 2006.

[63] B. Haskell, A. Puri, and A. Netravali, *Digital Video: An Introduction to MPEG-2*, ser. Digital Multimedia Standards. Kluwer Academic Publishers, 1997.

[64] K. Huang, I. Bacivarov, J. Liu, and W. Haid, "A Modular Fast Simulation Framework for Stream-Oriented MPSoC," in *IEEE Int'l Symposium on Industrial Embedded Systems (SIES)*, 2009, pp. 74–81.

[65] J. Castrillon, R. Velasquez, A. Stulova, W. Sheng, J. Ceng, R. Leupers, G. Ascheid, and H. Meyr, "Trace-Based KPN Composability Analysis for Mapping Simultaneous Applications to MPSoC Platforms," in *Proc. Design, Automation and Test in Europe (DATE)*, 2010, pp. 753–758.

[66] T. Isshiki, D. Li, H. Kunieda, T. Isomura, and K. Satou, "Trace-Driven Workload Simulation Method for Multiprocessor System-On-Chips," in *Proc. Annual Design Automation Conf. (DAC)*, 2009, pp. 232–237.

[67] C. Lee, S. Kim, and S. Ha, "A Systematic Design Space Exploration of MPSoC Based on Synchronous Data Flow Specification," *J. Signal. Process. Sys.*, vol. 58, pp. 193–213, 2010.

[68] H. Yang, S. Kim, and S. Ha, "An MILP-Based Performance Analysis Technique for Non-Preemptive Multitasking MPSoC," *IEEE T. Comput. Aid. D.*, vol. 29, no. 10, pp. 1600–1613, 2010.

[69] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-Aware Microarchitecture," in *Proc. Int'l Symposium on Computer Architecture (ISCA)*, Jun. 2003, pp. 2–13.

[70] S. Zhang and K. Chatha, "Approximation Algorithm for the Temperature-Aware Scheduling Problem," in *Proc. Int'l Conf. on Computer-Aided Design (ICCAD)*, Nov. 2007, pp. 281–288.

[71] S. Hafizovic and O. Paul, "Temperature-Dependent Thermal Conductivities of CMOS Layers by Micromachined Thermal Van Der Pauw Test Structures," *Sensors and Actuators A: Physical*, vol. 97–98, pp. 246–252, 2002.

[72] W. Haid, "Design and Performance Analysis of Multiprocessor Streaming Applications," Ph.D. dissertation, ETH Zurich, Oct. 2010.

[73] E. Lee and D. Messerschmitt, "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing," *IEEE T. Comput.*, vol. 36, no. 1, pp. 24–35, 1987.

[74] ——, "Synchronous Data Flow," *Proc. IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[75] A. H. Ghamarian, M. C. W. Geilen, T. Basten, and S. Stuijk, "Parametric Throughput Analysis of Synchronous Data Flow Graphs," in *Proc. Design, Automation and Test in Europe (DATE)*, 2008, pp. 116–121.

[76] B. D. Theelen, M. C. W. Geilen, T. Basten, J. P. M. Voeten, S. V. Gheorghita, and S. Stuijk, "A Scenario-Aware Data Flow Model for Combined Long-Run Average and Worst-Case Performance Analysis," in *Proc. ACM/IEEE Int'l Conf. on Formal Methods and Models for Co-Design (MEMOCODE)*, 2006, pp. 185–194.

[77] C. Brake, "Power Management In Portable ARM Based Systems," Accelent Systems, Tech. Rep., 2001.

[78] D. Tarjan, S. Thoziyoor, and N. Jouppi, "CACTI 4.0," HP Laboratories Palo-Alto, Tech. Rep. HPL-2006-86, 2006.

[79] S. Velusamy, W. Huang, J. Lach, M. Stan, and K. Skadron, "Monitoring Temperature in FPGA based SoCs," in *Proc. Int'l Conf. on Computer Design*, 2005, pp. 634–640.

[80] G. K. Wallace, "The JPEG Still Picture Compression Standard," *IEEE T. Consum. Electr.*, vol. 38, no. 1, 1992.

[81] J. T. Hsu and L. Vu-Quoc, "A Rational Formulation of Thermal Circuit Models for Electrothermal Simulation – Part II: Model Reduction Techniques," *IEEE T. Circ. and Sys. I: Fund. Theory and Applications*, vol. 43, no. 9, pp. 733–744, 1996.

[82] L. Thiele, S. Chakraborty, and M. Naedele, "Real-Time Calculus for Scheduling Hard Real-Time Systems," in *Proc. IEEE Int'l Symposium on Circuits and Systems (ISCAS)*, vol. 4, 2000, pp. 101–104.

[83] B. Friedland, *Control Systems Design: An Introduction to State-Space Methods.* Dover, 2005.

[84] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, ser. Lecture Notes In Computer Science. Springer, 2001, vol. 2050.

[85] N. Nise, *Control Systems Engineering*, 4th ed. John Wiley & Sons, 2004.

[86] G. Birkhoff and R. S. Varga, "Reactor Criticality and Nonnegative Matrices," *J. Soc. Ind. Appl. Math.*, vol. 6, no. 4, pp. 354–377, 1958.

[87] E. Wandeler, A. Maxiaguine, and L. Thiele, "Performance Analysis of Greedy Shapers in Real-Time Systems," in *Proc. Design, Automation and Test in Europe (DATE)*, 2006, pp. 444–449.

[88] S. Künzli, A. Hamann, R. Ernst, and L. Thiele, "Combined Approach to System Level Performance Analysis of Embedded Systems," in *Proc. IEEE/ACM Int'l Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2007, pp. 63–68.

[89] S. Chakraborty, Y. Liu, N. Stoimenov, L. Thiele, and E. Wandeler, "Interface-Based Rate Analysis of Embedded Systems," in *Proc. Int'l Real-Time Systems Symposium (RTSS)*, Dec. 2006, pp. 25–34.

[90] J. Cutts, "A Computer in Every Pocket, 4G in Every Town," in *5 Technology Trends to Watch.* Consumer Electronics Association (CEA), 2010.

[91] *The Benefits of Multiple CPU Cores in Mobile Devices*, NVIDIA, 2010, Whitepaper.

[92] W. Haid, K. Huang, and S. Künzli, "DOL: Application Examples," Jul. 2008, revision 1069.