UNIVERSITÄT PADERBORN
*Die Universität der Informationsgesellschaft*

# Multi-aspect Full-system Server Model and Optimization Concept as a Simulation-based Approach (MFSMOS)

**Dissertation**

in Computer Science

A thesis submitted to the

**Faculty of Computer Science, Electrical Engineering and Mathematics, Paderborn University**

in partial fulfillment of the requirements for the degree of

***doctor rerum naturalium***

**(Dr. rer. nat.)**

by

**Diana Riemer**

Paderborn, Germany, 2017-11-29

*Supervisors:*

|  |  |
|---|---|
| Prof. Dr.-Ing. André Brinkmann | Johannes Gutenberg University Mainz |
| Prof. Dr. Franz Josef Rammig | Paderborn University |

# Abstract

The exponential growth of the information technology, such as social media content or on-demand video services, results in increasing energy consumption, which may be one of the main causes of climate changes. Power and cooling are the key challenges to reducing the emission of greenhouse gas, especially in large-scale data centers. With our full-system model, we contribute towards improving the energy efficiency on specific server systems and, consequently, optimize the efficiency of data center infrastructures. We develop a generic, flexible, and scalable model to simulate and optimize a complete server system for the multiple, potentially conflicting aspects: power, temperature, and performance. We develop a hierarchical and abstract model of a rack-mounted server system, which builds the base of our mathematical methods to calculate the multi-aspects of each component. We demonstrate the feasibility and advantages of our concept through a prototypical implementation, in which we empirically validate our model using a variety of artificial workloads to ensure the reproducibility at any time. In principle, our simulation-based, full-system server model supports customer-specific workload scenarios, specified as realistic category-specific utilization levels, to simulate the suitable power of server systems. We address the significant static as well as dynamic characteristics and configurations to cover a variety of server systems and components compatible with the customer-specific server system. We precisely calculate the power consumption that reduces the over-provisioning of the server system, particularly in industrial practice. Moreover, we demonstrate that we can forecast future generations of high-performance systems and components by assuming the predecessor or a similar generation. To our knowledge, in academic research, there are no generic approaches that cover the full server system simulation on a common base. This thesis provides new research contributions that explicitly cover the heterogeneous characteristics of the hardware and software variations, such as supporting diverse server families or generations. Moreover, the simulation optimizes the energy efficiency of the server system at various utilization levels, especially at low-intensity phases (under-utilization).

## Zusammenfassung

Der ansteigende Medienkonsum, bedingt durch sich weiterentwickelnde Kommunikationskanäle, führt zum kontinuierlichen Anstieg des Energiebedarfs, welches den Klimawandel weiter voranschreiten lässt. Das größte Potenzial zur Energieeinsparung im Rechenzentrum wird derzeit dem IT-Equipment sowie dessen Kühlung zugesprochen. Wir haben ein ganzheitliches Systemmodell zur Optimierung der Energieeffizienz von Rack-Servern entwickelt. Wir entwickelten ein allgemein-gültiges, flexibles und skalierbares Model, um verschiedene Server zu simulieren und optimieren. Das hierarchische und abstrahierte Servermodell unterstützt die Berechnung der konkurrierenden Aspekte: Energieverbrauch, Temperaturentwicklung und Performance. Die prototypische Implementierung zeigt die Machbarkeit und Vorteile unseres Ansatzes. In der Evaluation unseres simulationsbasierten Systemmodells verwenden wir synthetische Lastszenarien zur besseren Nachvollziehbarkeit, wobei wir auch realistische benutzerspezifische Lastszenarien unterstützen. Der Anwender definiert die prozentuale Auslastung der Komponenten als Lastszenario, welches wir zur Berechnung der maximalen Leistungsaufnahme und des Energieverbrauchs verwenden. Wir präsentieren die relevanten statischen und dynamischen Merkmale, um unterschiedliche Serversysteme und Komponenten aus verschiedenen Generationen abzubilden. Wir berechnen den Energieverbrauch bzw. die Leistungsaufnahme und die daraus resultierenden Temperaturen hinreichend genau, welches die oftmals erhebliche Überschätzung des tatsächlichen Energieverbrauchs von Servern reduziert. Unser Ansatz ermöglicht die Prognose der maximalen Leistungsaufnahme von zukünftigen Systemen und Komponenten. Unser Ansatz unterstützt, im Gegensatz zu den bisherigen akademischen Ansätzen, eine Vielzahl an Server und deren unterschiedliche Komponenten. Mithilfe unseres simulationsbasierten Ansatzes können wir die Energieeffizienz von Servern bei jeglichen anwenderspezifischen Szenarien optimieren.

# Acknowledgement

First and foremost, I would like to express my deep appreciation to Professor Dr.-Ing. André Brinkmann and Professor Dr. Franz J. Rammig, my research supervisors, for their patient guidance, vital support, immense knowledge, and constructive suggestions during the planning and development of this dissertation. I could never have imagined having better mentors for my PhD study. In particular, I sincerely thank Franz for his extreme patience in the face of numerous obstacles and his invaluable advice towards the success of my thesis. Besides my advisors, I would like to thank my committee members, Professor Dr. Christian Plessl, Professor Dr. Gudrun Oevel, and Dr. Wolfgang Müller for their time, insightful comments, and hard questions in the examination board. I would like to pay special thanks to Cornelia Wiederhold who was periodically busy with printing, binding and delivering this thesis.

I would also like to express my very profound gratitude to Dr. Wolfgang Müller and Bernhard Homölle, who initiated the cooperation between C-LAB at Paderborn University and Fujitsu Technology Solutions GmbH. My regards go out to the management of both persons for the third-party project and for the contributions, that each of them made to my intellectual growth during my years of study. I would like to thank the following staffers at Fujitsu Technology Solutions GmbH as well as Atos IT Solutions and Services GmbH: Karsten Beins, Hansfried Block, Dr. Martin Clemens, Thomas Diekmann, Helmut Emmerich, Wolfgang Erig, Stefan Festl, Robert Hoffmann, Sebastian Keßler, Brigitte Krawinkel, Alois Lichtenstern, Johannes Linne, Bernhard Lüthen, Karl-Josef Lüttgenau, Uwe Meiners, Georg Müller, Dr. Oliver Niehörster, Detlef Pusch, Klaus Rammig, Dr. Mitsuru Sato, Bernhard Schräder, Dr. Jürgen Schrage, Jothiram Selvam, Dieter Tenberg, Sylvia Tuschen, Walter Unruh, Bernd Winkelsträter, and Norbert Zobe. You provided enormous support and were always willing to help me. I must express my very great appreciation to Harald Lüdtke for providing me valuable support with *Kalcheck*, immense knowledge, and continuous motivation throughout the ongoing process of researching and writing this thesis. I would also like to extend my thanks to the technicians of the environmental laboratory and hardware department at Fujitsu Technology Solutions GmbH for their technical support, such as collecting the server-specific data, providing the necessary IT equipment, and introducing me to the tools for my measurements and analysis.

This dissertation was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre On-The-Fly Computing (SFB901). My special thanks are extended to Dr. Michelle R. Kloppenburg, who assisted through proofreading, my English teacher Mary Cremer, and classmates Annika Ballhausen, Michaela Kemper, Kirstin Köhler, Birgit Petermeier, and Annette Zaloudek for enhancing my speaking language skills.

Furthermore, I would like to show my gratitude to my fellow doctoral students Peer Adelt, Markus Becker, Dr. Gilles Bertrand Gnokam Defo, Jan Jatzkowski, Mabel Mary Joy, Dr. Alexander Jungmann, Dr. Matthias Keller, Christoph Kuznik, Fabian Mischkalla, Sven Schönberg, Dr. Katharina Stahl, and Jan Stenner for their feedback, conversations, and of course friendship. I would like to offer my special thanks to Dr. Lorijn van Rooijen for her guidance in defining precise mathematical formulas. Getting through my studies required more than academic support and I would like to offer my special thanks to Jan Braun, Claudia Dobrinski, Marlies Drestomark, Saikal Gerlach, Isabel Junger, Andreas Kaufmann, Dr. Isabel Koke, Christoph Nagel, and Dr. Gaby Nordendorf for listening to me, tolerating me, and supporting me over the last several years. Indeed, it would have been a lonely office without Veit Dornseifer, Mohamed Hassan Mohamed Elsharkawi, Bastian Koppelmann, Arjun Ramaswami, and Jonathan Westerholt. Finally, I especially thank Ginger Claassen for our daily conversations and your morning-based reliable coffee delivery.

I would also like to thank my friends for their wise counsel and infinite support. Furthermore, I am particularly grateful for my family and, especially, my parents who raised me with a tremendous passion for technology, particularly in computer science, and supported me in the pursuit of this project. They were always supporting me and encouraging me with their best wishes. I would like to thank Bernd Rummler very much for his widespread support, giving me confidence, and always being there for me.

Finally, I would like to thank all those mentors, advisors and friends who have supported me and believed in me. Just wishing to recognize the valuable help from all throughout my PhD thesis - I ask for forgiveness for all those who have been with me over these years whose names I have failed to mention.

Thank you,
Diana Riemer
Paderborn University
November 2017

# Table of Contents

# 1 Introduction

## 1.1 Motivation

The consumption rate of information technologies, especially web-based resources, has increased rapidly. Social media networks, such as Facebook and Twitter are well established. Providers work with large amounts of data because the customers upload and download a wide range of data, such as movies, photos, and documents. Every day consumers upload more than 300 million photos on Facebook (data from the first quarter of 2012) and thus, data centers daily handle more than 500 terabytes of new data [Fac 2012, Con 2012]. The *International Data Corporation*[1] (IDC) forecasts the amounts of information will double every two years, as shown in Figure 1 [JD 2012, Pet 2012]. The amount of data was nearly five Zettabytes[2] in 2013 and will probably be around 40 Zettabytes in 2020, which shows exponential growth.

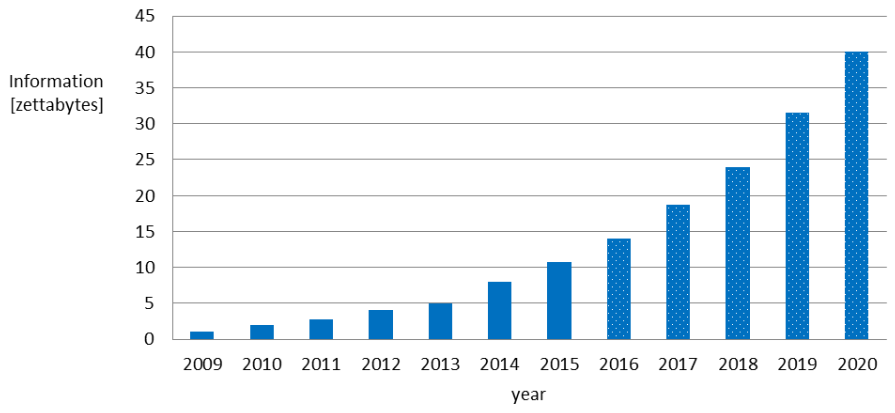


Figure 1: IDC's forecast for the digital information growth [JD 2012]

This growing data trend is occurring because of different factors of mobile devices, forecasted by Gartner 2012 [Pet 2012]. The shift from mobile phones to smartphones provides the ability to perform computer functions on the portable devices. Therefore, some general-purpose services, such as television, are switching to on-demand video services. Wide ranges of business-to-consumer (B2C) markets are growing because of the web-based streams. Especially the marketplace for portable applications has grown within the last years. The authors in [Pet 2012] forecasted nearly 70 billion mobile application downloads in 2014.

The demands of computation resources, not just storage resources, are also increasing. Data centers or search engine providers such as *Bing*[3] or *Google*[4] handle a high level of data quantity, inquiries, and services using the server systems. Google had approximately sixty

[1] International Data Corporation: http://www.idc.com
[2] Zettabyte: $10^{21}$ Byte
[3] Bing: http://www.bing.com
[4] Google: https://www.google.com

thousand searches per second in 2012 [Com 2013]. If we take previous years into account, the amount of searches will dramatically increase in the following years. In 2010, Google's server consumed nearly two billion kilowatt-hours [Koo 2011]. As a consequence, the energy demand on data centers is growing.

The energy efficiency report produced by the *US Environmental Protection Agency*[5] (EPA) depicts historical as well as current trends of energy use in U.S. data centers [USEPA 2013]. Within six years the energy consumption doubled between 2000 and 2006 from 30 to 60 terawatt-hours[6], see Figure 2. Assuming the trend is continuing, the energy use will be approximately 170 terawatt-hours in 2020. It has thus increased more than fourfold [Acc 2008].

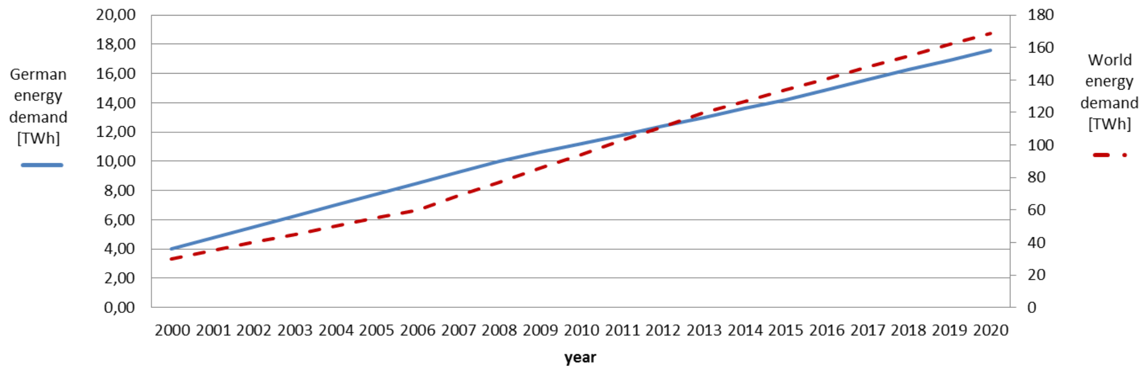


Figure 2: Total energy demand of servers and data centers [Acc 2008, HFS 2010]

The energy demand of German data centers was 12 terawatt-hours in 2012 [Thy 2012]. The servers and data centers require about 1.8 percent of the overall German energy demand. Another agency [HFS 2010] forecasted that in 2015 the energy demand of servers and data centers in Germany would be around 14.2 terawatt-hours in "business as usual cases" and around six terawatt-hours in "best practice" green IT-based solutions, as shown in Figure 3.

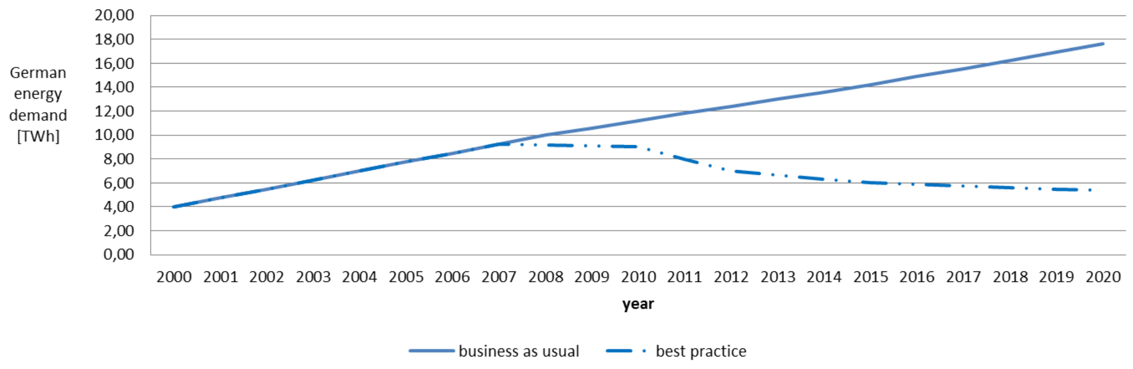


Figure 3: Total energy demand of German data centers [HFS 2010, Thy 2012]

[5] US Environmental Protection Agency: http://www.epa.gov

[6] Terawatt: $10^{12}$ Watt

Furthermore, Koomey forecasts the same tendency for computer performance (operations per period) and efficiency (operations per power demand), which double within one and a half-year [KBS et al. 2010]. The costs per operation are decreasing because of the technologies that are shrinking semiconductors and because of increasing transistor density. In 1965, Moore predicted a doubled numbers of transistors on integrated circuits every two years [Moo 1965]. An adequate computer can be built much cheaper with the same performance, but using more transistors. An increasing computing performance enables large-scale computations in academia as well. For instance, transmutation, decoding, reconstructing, or recombining of genomes generates large data sets and complex tasks. These data-intensive tasks utilize all data center levels [IIZ et al. 2007], especially the server, in certain ways.

The authors in [BLR et al. 2005, BH 2007, RRT et al. 2008, LEU et al. 2010] state that the servers consume the most power (between $27\%$ and $65\%$) in comparison to the data centers' operational expenditures, such as the overall electricity costs. The increasing energy consumption may be one of the main causes of climate change. Power and cooling are the key challenges to reducing especially the greenhouse gas emissions of data centers. Several governmental agencies, initiatives, and industrial consortiums are investigating on improving the energy efficiency. Replacing inefficient IT equipment, optimizing the climate control, limiting, and balancing the server workloads are conventional considerations for reducing the energy consumption. With our full-system model, we want to make our contribution to enable certain energy efficiency optimizations on specific server systems. We model a complete server system as a simulation-based approach to predict the power consumption, temperature, and performance, which enables the analysis of the energy efficiency considering realistic workloads and low-intensity phases (under-utilization). Moreover, we optimize the energy efficiency of the server system concerning the variety of software and hardware configurations.

## 1.2 Objectives and Contributions

The main objective of this thesis is to develop a novel multi-aspect full-system model that is able to optimize an entire server system concerning the energy efficiency. We specify a flexible simulation-based approach that enables the optimization of customer-specific workload scenarios and various system configurations. In this thesis, we investigate the following questions:

- How can we specify entire server systems and their components?

  *We specify a server-specific configuration tree as a hierarchical structure concerning the various abstraction levels. We subdivide the hardware, first, into the static model considering the architecture as well as connectors, and secondly into the dynamic behavior model, including the relations between the components.*

- Is it possible to forecast next-generation systems?

  *We specify various significant characteristics to forecast the power consumption of next-generation processors and memory modules that are based on the component technology development, for instance.*

- How can we deal with the external and internal requirements concerning the workload scenario to support vendor and customer demands?

  *We flexibly specify the server system externals, such as the resource-based workload scenarios to apply the intended scope of application. We integrate the internal and external constraints such as the thermal limits that build the base of the thermal control in the primary phase of our optimization strategy.*

- How can we calculate aspects of power, temperature, and performance in an accurate manner?

  *For each aspect, we develop a method that consists of the technical specification function and the configuration function. In our configuration tree, we weight the characteristics with their corresponding coefficients and consider the related offsets.*

- How can we deal with multi-aspect-based calculation methods and their relationships, especially the interdependencies between the components?

  *We implement each aspect in a particular calculation method. Herein, the power consumption builds the significant input parameter of the thermal calculation. We predict the performance on the basis of the power consumption and thermal development. We realize the interdependencies between the components in Simulink and in MATLAB partly specify the relations that influence the component behavior.*

- What are the significant characteristics of the components concerning the specific aspects?

  *In contrast to common assumptions and as a result of our analysis, we found the following significant characteristics of the memory modules: vendor, die, series, fabrication size, synchronization mode, and ranks. We additionally observe the relevant processor characteristics: semiconductor technology (thermal design power), product life cycle stage, fabrication size, and series. The power consumption, thermal development, and performance of the server system depend upon the enclosure, its subset of equipment, and usage models.*

- How can we flexibly react to characterization changes and adjust our calculation methods?

  *We define a centralized database that consists of the possible characteristics and configurations. We provide access to individually configurable data within the database to enable the use of our models across multiple server generations.*

- Is it possible to simulate a full-server system in a time-continuous workload scenario?

  *Indeed, we simulate the entire server system concerning the trace of several real application softwares. In our evaluation, we trace the utilization levels from artificial benchmarks to ensure the reproducibility at any time.*

- How can we deal with multi-objective optimization of the entire server system?

  *In our evaluation, we present the alternations of the server characteristics and configurations at each time $t_k$ resulting from our multi-objective optimization in which we select the global optimal solution.*

- How much can we optimize the energy efficiency by adjusting a more suitable configuration or characteristic?

  *We illustrate the possible optimization of the performance-to-power ratio by nearly* $12.2\%$ *in our exemplary evaluation.*

- How much amount of power consumption does an improved server system (configuration or characteristic) accomplish regarding a specific workload scenario?

  *We exemplarily achieve the mean processor power reduction of approximately* $53.3\%$ *when analyzing the SPECpower benchmark.*

Our multi-aspect full-system model should minimize the vendors' measurement effort by simulating accurate and precise aspects of the server system components. As a result, we will reduce the over-provisioning of the server system. Moreover, we investigate the opportunity to calculate next-generation systems while considering actual trends of server system configuration.

## 1.3 Organization of This Thesis

We organize the remaining content of this thesis as follows:

**Chapter 2 – Background** outlines the fundamental knowledge to understand our modeling and simulation. Herein, we define necessary terms pertaining to energy efficiency and give the backgrounds of the server system in a data center.

**Chapter 3 – Basic Modeling Technologies, Algorithms, and Approaches in Academic Research and Industrial Practice** presents the various modeling technologies and respective server system domains to create multi-aspect models. Moreover, we introduce the corresponding related work, which we split into aspect-based sections. We describe the gap between the academic approaches and the industrial field of application, which results in our problems.

**Chapter 4 – Problem Statement, Challenges, and Aims** discusses the problems and challenges between the industrial tools and academic approaches. For this, we specify seven aims used as a basis for our thesis.

**Chapter 5 – Multi-aspect Full-system Server Model and Optimization Concept as a Simulation-based Approach (MFSMOS)** presents the details of the five-step concept, including the aspect-based component models and characterization of the server system. We describe the external environment and specify further details of the optimization strategy, such as the cascading primary and secondary phases.

**Chapter 6 – Design and Implementation of the Architecture** applies the concept in a simulation framework as a prototypical implementation. We realize a *Model-View-Controller* (MVC) approach and describe the layers concerning its methods.

**Chapter 7 – Evaluation of the Multi-aspect Full-system Server Model and Optimization (MFSMOS)** evaluates the multi-aspect-based methods and algorithms developed in this thesis. First, we present the evaluation environment and analyze the calculation methods regarding their accuracy. Second, we investigate the impact on changes of the component characteristics. Third, we evaluate the improvements that result through our server system optimization.

**Chapter 8 – Conclusion** summarizes the work done in this thesis and presents an outlook in which further investigations might be useful.

# 2 Background

The purpose of a model is to show and predict realistic behavior, which interacts with the environment. Thus principles[7] and theories are formal logic or mathematics, for instance. Real behavior is defined in an abstract and generalized manner. Models represent certain aspects of reality with a specific purpose. They support the understanding of relationships in the case of similar behavior in comparison to the real world. Section 2.1 describes the modeling and simulation differences. This thesis, which addresses an energy efficiency estimation approach, contains terms such as power, energy, performance, and efficiency, which are outlined in detail in Section 2.2. The purpose of this model is to predict the power and energy consumption of server systems in data centers. Sections 2.3 and 2.4 present a generic overview of the services and equipment of data centers.

## 2.1 Modeling and Simulation

Modeling is an approach to show and predict an actual system’s reality. A model depicts a simplified representation abstracting from some real-life complexities. The model accuracy depends on the level of the model’s detail. Too many details result in complex, complicated and time-consuming models. On the other hand, providing too little information will have the effect of missing relevant details for the simulation results. Therefore, a specification and requirement analysis are mandatory steps for developing a proper model. A model is a virtual or digital prototype of a real system dependent on use cases and model objectives. Modeling systems need system details. If customers know the internal rules or workflows, they use a white-box modeling approach for the respective system. This is the case in self-designed systems. On the other hand, modeling a black-box does not include any internal details of the system. Such an approach is required in case of external systems from other suppliers. In contrast, a grey-box model is a mixture of both the white-box and black-box approach. Some internal workflows are known but not at all of them.

A model is created by empirical as well as mathematical methods and techniques such as deterministic, stochastic, static, and dynamic methods. Usually, equations, logical rules, and constraints define the limits, and flowcharts represent the system behavior. Stimulators generate input data for the model. In an accurate case, the system model is an exact replica of the real system with the same behavior. The developer executes the model to check whether the right system [BCC et al. 2014] is built.

Simulations of the model help to verify and confirm the represented reality[8] and dynamic behavior. Furthermore, they support the processes of analyzing and designing a system. Under known input conditions, a model is valid if the resulting outputs from the real world and

[7] Principles: physical, analogue, or mathematical model
[8] Reality: logical, behavioral flow, interfaces, or triggers

simulation are the same. The decision depends on the accuracy and precision of the system. Simulation-based approaches handle the complexity of industrial hardware and software systems [SMA et al. 2003]. A simulation checks a model, which runs over a certain period. A positive aspect is that inside a simulation all operating conditions such as temperatures, as well as non-controllable factors, e.g. the weather, are changeable. It does not matter if it is realistic or not. Sometimes arbitrary conditions are dangerous or expensive in the real world. On the other side, a simulation requires a detailed model to predict valid behavior and results.

Either a suitable programming language or simulation package is required for implementing a simulation model. Choosing the adequate simulation software or environment depends on the software or customer properties. Software properties include the following aspects:

- Support
- Documentation
- Interfaces
- Costs
- Resource requirements
- Statistical capabilities
- Reporting capabilities

On the other hand, customer properties are potential effects, problems, aid for structures (hierarchical, flat, object-oriented, or nested) and their level of competence or expertise because of training periods.

### 2.1.1 MATLAB and Simulink

MathWorks[9] developed a software system called MATLAB. MATLAB is an advanced tool for numerical computations, used in academia and industry. MATLAB is an environment and a programming language. It focuses on vector-, and matrix-based calculations. The model used in this thesis includes several differential equations, statistics, and forecasts. MATLAB solves these problems and provides various interpolation methods as well as statistical analysis. For this thesis, we analyze and integrate the high level of data within the model. MATLAB supports several import functions and opportunities. A wizard-based graphical interface generates a MATLAB function based on several data, such as input information or measurement results. Furthermore, the data can be adapted, analyzed (reports), and visualized (plots) by built-in or plug-in tools. Moreover, many domains are used: for instance, control, sequence, mechanical, or electrical systems. MATLAB supports model, data, and controller optimization. Automatic code generation is another benefit. In this thesis, a MATLAB extension called Simulink is used.

[9] MathWorks: http://www.mathworks.com

Simulink is a block-based simulation environment with a graphical user interface and is integrated into the MATLAB environment. Its interfaces support executing MATLAB code (M scripts) and functions. Many hierarchical domains are distinguished in blocks, which are also called subsystems. This thesis uses a model-based design (MBD) approach, which models the control system and simulates the dynamic behavior of electrical and mechanical systems. Moreover, Simulink supports the textual modeling of methods, data flow diagrams, state machines, and various domains. A simulation model includes, for instance, the dynamic control of non-deterministic behavior of disturbances, influences, and environmental factors. Simulink supports viewing and debugging results for model optimization as well as parameterization and additionally supports executable source code generation.

### 2.1.2 MATLAB Notation and Syntax

The following chapter introduces MATLAB notations. The model uses vectors, matrices, and arrays to define the represented system. Developed algorithms and system descriptions use MATLAB syntax as well.

MATLAB labels variables with an equal sign. The variable name is on the left side, whereas the variable content is on the right side. Appendix A3c describes MATLAB label restrictions. MATLAB variables contain only numbers $(I)$ or a mix of numbers and strings $(II)$. The mix is a cell array or cell matrix. The following terms describe the syntax of MATLAB vectors or matrices. A numerical vector $(I)$ consists of a sequence of numbers within square brackets as shown in (2.1). A number is followed by a number within a row vector. Each number defines a column, whereby a blank separates the columns.

$$[label] = [\ [numbers]^{+}\ [numbers]^{*}\ ] \tag{2.1}$$

A row vector transforms into a column vector using a semicolon as shown in (2.2). A semicolon defines the end of a row.

$$[\mathrm{label}] = [\ [\mathrm{numbers}]^{+};\ [\mathrm{numbers}]^{+}\ ] \tag{2.2}$$

Both variable types $I, II$ are within square brackets. A cell array $(II)$ requires additionally curly brackets and single quotation marks as shown in (2.3, 2.4). Strings and numbers are in any order.

$$[\mathrm{label}] = [\{\ [{}'[\mathrm{string|numbers}]{}']^{+}\ \ [{}'[\mathrm{string|numbers}]{}']^{*}\}\ ] \qquad (2.3,\ 2.4)$$

$$[\mathrm{label}] = [\ \{\ [{}'[\mathrm{string|numbers}]{}']^{+}\ [{}'[\mathrm{string|numbers}]{}']^{*}\ ;$$

$$\{\ [{}'[\mathbf{string|numbers}]{}']^{+}\ [{}'[\mathbf{string|numbers}]{}']^{*}\ \}\ ]$$

The following examples show both matrix types $I, II$. The matrix $Ma$ includes many vectors. The matrix dimension $(m\ x\ n)$ specifies the amount of rows $(m)$ and columns $(n)$. A complete numerical matrix $Ma\ (I)$ is shown in (2.5). It can contain different dimensions, for instance $(2x2, 2x3, 3x2)$.

```
Ma=[1 2; 3 4]  Ma=[1 2 3; 4 5 6]  Ma=[1 2; 3 4; 5 6]
%   1 2                 1 2 3              1 2
%   3 4                 4 5 6              3 4
%                                          5 6
```
(2.5)

An index identifies a specific value from a vector or matrix. The first mandatory parameter determines the column of a vector and a row in the case of matrices. In matrices, the second parameter additionally identifies the column. A comma separates both values, as shown in (2.6).

```
Ma=[1 2 3; 4 5 6]       Ma(1,3)         Ma(2,3)
% 1 2 3                 3               6
% 4 5 6
```
(2.6)

The same matrix $Ma$ uses mixed data types $(II)$ in the following example (2.7). The values *two* and *four* are numbers in both definitions. The other values are converted to strings in $Ma\_cell\_array$. Therefore, we use single equation marks within the matrix.

```
Ma_cell_array = [{'one' '2' 'three'; '4' 'five' 'six'}]
%      'one'      '2'        'three'
%      '4'        'five'      'six'
```
(2.7)

A data adaption or extension is manageable by changing or adding new values independently of their data types. MATLAB provides a wide range of cell-specific functions, such as converting strings to numbers $str2num$, and common functionalities. Further details about the MATLAB notation and syntax are given by the MATLAB homepage[10] and Appendix A3c. The simulation model of this thesis uses MATLAB, its functions and variables. Therefore, definitions are provided in MATLAB notation. The model of this thesis focuses on power dissipation and energy efficiency. The following section defines power- and energy-related terms.

## 2.2 Definition of Terms

### 2.2.1 Power and Energy

Power $P$ is an electrical level in watt $[W]$ and exists at any certain point in time [Stö 2014] of a defined duration $T$, see Equation (2.8). Power is the average value of power oscillation. At each point in time, the simulation system models a certain state with a specific power consumption. This thesis focuses on peak power consumption, which is calculated from the first derivative of the power function. Over all points in time, we look for the largest power

[10] MATLAB homepage: MathWorks, http://www.mathworks.com/help/index.html

demand. This process is time-invariant and uses those values. Power in DC circuits[11] is found through Joule's Law and is defined in Equation (2.10) for instantaneous values. Additionally, peak values are taken instead of *rms*-based[12] values. As a result, power is calculated as a product of voltage and current, see (2.11). Power consumption, dissipation, and demand have the same significance.

**Equation 1: Power calculation**

$$P = \frac{1}{T}\int_0^T v(t) * i(t)\, dt \tag{2.8}$$

$$P(t) = V(t) * I(t) \tag{2.9}$$

$$P = V * I = I^2 * R \tag{2.10}$$

$$P[W] = V[V] * I[A] \tag{2.11}$$

Concerning the power of IT equipment, this is distinguished between peak and nameplate power. Peak power is the amplitude of power oscillation in DC circuits plus the largest power in AC circuits[13]. Nameplate power is the nominal power of electric power production equipment. A special system-rating label (nameplate) contains information about how much power the equipment consumes. It is necessary for electrical installations to select the right wiring method to meet these requirements.

Integrating electrical power consumption over a period of time $T = t_2 - t_1$ results in electrical energy $E$. The commercial unit of energy is a watt-hour $[Wh]$ or Joule $[J = Nm/s]$, whereby Joule is defined as newton meter per second. Equation (2.12) changes into a time-invariant approach. Therefore, only time $T[s]$ is multiplied with power values $P[W]$, see (2.14).

**Equation 2: Energy calculation**

$$E = \int_{t_1}^{t_2} p(t)\, dt = \int_{t_1}^{t_2} i(t) * v(t)\, dt \tag{2.12}$$

$$E = P * (t_2 - t_1) = V * I * (t_2 - t_1) \tag{2.13}$$

$$E[Wh] = P[W] * T[h] = V[V] * I[A] * T[h] \tag{2.14}$$

In the following example, the power at time 15 minutes is 80 watts, see dotted lines. Twenty minutes later the (peak) power doubles to 160 watts, shown with dashed lines. The median power demand is about 122 watts. The colored grid area below the curve is equal to the

[11] DC circuits: direct current circuits
[12] Rms-based values: root-mean-square
[13] AC circuits: alternating current circuits

overall energy consumption within a period. Adding up power values up to an hour, beginning at the first minute, results in an energy demand of approximately 7.4 kilowatt hours ($kWh$).

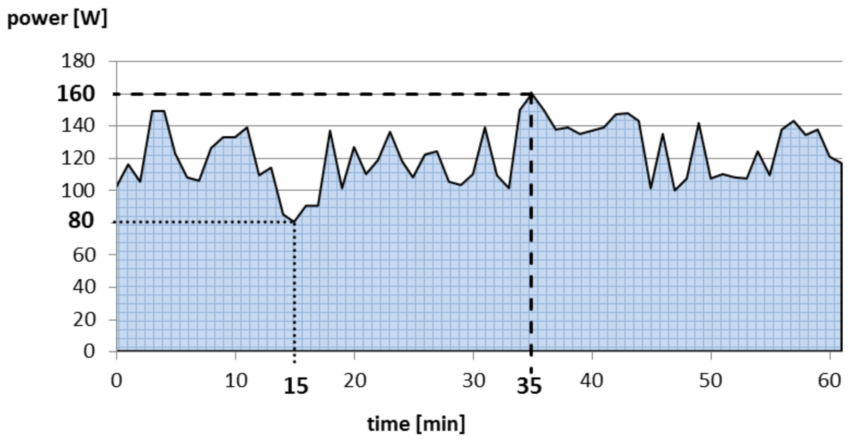


Figure 4: Power $[W]$ vs. energy $[Wh]$

### 2.2.2 Performance

Performance[14] indicates the effectiveness of a system. It is the ratio between the (execution) time and the resources consumed for a given task or a set of operations (computations). More aspects, such as the delay of processing, access, transmission, resource usage, or response time (latency) as well as throughput, are relevant for various measurements. Performance is specific to an application and the used system. Cycles per instruction (CPI) and cycle times differ between processor types because of architecture variations, such as the cache size and level. Performance counters may include device cycles, cache misses/hits, or a number of instructions. Furthermore, an executed application utilizes resources in different ways. Therefore, performance criteria depend on the system specifics. Usually, benchmarks measure the performance. The Standard Performance Evaluation Corporation (SPEC)[15] developed well-known benchmarks. A benchmark is a suite containing different operation types. Floating-point and integer operations are common parts of processor testing. In general, a higher performance score determines better-used system resources. Processor performance usually is measured in floating-point operations per second (FLOPS) or in response / transaction time, which is defined as required time to finish a job.

### 2.2.3 Utilization and Workload

Utilization[16] is the percentage of component usage in relation to its maximal available physical working capacity. It is the ratio between working (active) and idle time. The peak utilization $(100\%)$ uses the entire possible working capacity that a resource offers. In this case, the system or a component is 100-percent busy processing during a given interval and the capacity

[14] Performance: performance factor
[15] SPEC: http://www.spec.org/
[16] Utilization: sometimes called load level

limit is reached. Utilization levels distinguish between static component states. Utilization $U$ is defined as a utilization level $u_m$ at a specific point in time $t_i$ for each component $m$ of a system, whereby $m$ is any natural number $m, n \in N_0, N_0 = \{0, 1, 2, 3, \ldots\}$. A column vector $\vec{U}$ specifies system utilization at a specific time, as shown in Equation (2.16). The time vector $T$ contains $n$ values $t_n$. If $n$ is infinite, the time will be continuous. The vector $\overrightarrow{u_1}$ specifies the utilization of $m$ components at $t_1$. A tuple $(t_1, u_{1\ldots m})$ specifies a system utilization only for one point in time and is herein defined as $\overrightarrow{u_1}$, which is furthermore done for all $n$ values of $T$, see (2.23).

$$\vec{U} = \{u_1, u_2, \ldots, u_m\} \text{ at } t_i, \forall i \in N_0, m \in N_0 \tag{2.15}$$

$$\vec{U} = \begin{pmatrix} u_1 \\ u_2 \\ \ldots \\ u_m \end{pmatrix} \text{ at } t_i, \forall i \in N_0, m \in N_0 \tag{2.16}$$

$$T = \{t_1, t_2, \ldots, t_n\} \tag{2.17}$$

$$\overrightarrow{u_1} = \{u_{11}, u_{21}, \ldots, u_{m1}\} \text{ at } t_1,\ m \in N_0 \tag{2.18}$$

$$\overrightarrow{u_1} \equiv (t_1), \begin{pmatrix} u_{11} \\ u_{21} \\ \vdots \\ u_{m1} \end{pmatrix} \tag{2.19}$$

$$\overrightarrow{u_2} = \{u_{12}, u_{22}, \ldots, u_{m2}\} \text{ at } t_2,\ m \in N_0 \tag{2.20}$$

$$\overrightarrow{u_2} \equiv (t_2), \begin{pmatrix} u_{12} \\ u_{22} \\ \vdots \\ u_{m2} \end{pmatrix} \tag{2.21}$$

$$\overrightarrow{u_m} = \{u_{1n}, u_{2n}, \ldots, u_{mn}\} \text{ at } t_m,\ m \in N_0 \tag{2.22}$$

$$\overrightarrow{u_m} \equiv (t_n), \begin{pmatrix} u_{1n} \\ u_{2n} \\ \vdots \\ u_{mn} \end{pmatrix} \tag{2.23}$$

Workload is a specific application[17] with many processes that are running or executed on the system. A workload distinguishes between different utilization levels $U$ of each component $m$ at a specific time $t_n$, varying with the executed applications. Workload $W$ is an $(m\ x\ n)$ matrix, which unfolds when the point in time $t_n$ is defined as the $n$-th column vector of the matrix. The workload matrix $W$ reflects a period of time $T$, see Equation (2.24). The columns $n$ in the matrix $W$ represent the time index. Furthermore, the dimension $m$ of the matrix corresponds to the number of components $m$ and is scalable just like the system.

[17] Application: operating system, program, or software

$$W = (w_{ij}) = \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ u_{21} & u_{22} & \dots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ u_{m1} & u_{m2} & \dots & u_{mn} \end{pmatrix} \forall i \in N_0,\ \forall j \in N_0,\ m,n \in N_0 \qquad (2.24)$$

$$N_0 = \{0, 1, 2, 3, \dots\}$$

A workload scenario is a specific workload or concatenated workload profiles running on the system over a period of time.

### 2.2.4 Thermal Energy, Temperature, and Heat

The thermal energy is the total of all kinetic energies within a system, which is part of the particle's movement or motion, such as for heating up an object. The unit of thermal energy is Joule $[J]$, which has an internal temperature and produces heat. The temperature is an absolute internal energy value of the resource's state, which refers to the average kinetic energy. We measure it in degrees Celsius $[°C]$, Kelvin $[K]$, or Fahrenheit $[°F]$. The heat (internal energy) is the transfer of thermal energy from one system to another. It is the transfer between two objects of different temperatures or the same system flowing from one temperature to another. The heat is caused by the flow of thermal energy and is measured in Joules $[J]$ or watt-hours $[Wh]$. The well-known types of heat transfer are conduction, convection, and radiation. The thermal energy is directly proportional to the temperature.

The analogy between the electric and thermal quantities offers an alternative description of the system. The thermal balance follows the same rules compared to Ohm's law of the conservation of electrical energy, which is shown in Table 1. The classical thermal properties are current and thermal resistance[18].

Table 1: Electrical and thermal analogy

| Electric quantity | Thermal quantity |
|---|---|
| $I\ [A]$ Current flowing through | $q\ [W]$ Rate of heat conduction, heat flow |
| $\Delta V\ [V]$ Voltage | $\Delta T\ [K]$ Temperature difference |
| $R_e\ [\Omega]$ Electrical resistance | $R_t\ [K/W]$ Thermal resistance |
| $C\ [F]$ Electrical capacitance | $C_t\ [J/K]$ Thermal capacitance, thermal mass |
| $\tau = RC\ [s]$ Electrical constant | $\tau_t = R_t C_t\ [s]$ Thermal constant |
| $I = \frac{\Delta V}{R_e}$ | $q = \frac{\Delta T}{R_t}$ |

[18] Thermal resistance: theta (θ) is a characteristic of a heat sink or the specific thermal resistance is a material constant [Int 2015]

### 2.2.5 Efficiency

Efficiency distinguishes between thermal, energy, working, computational, and economic efficiency. In the context of this work, computational energy efficiency is always used. Efficiency $\eta$ is the ratio between outputs $y_i$, and inputs $x_i$, being a unit-less ratio, see Equation (2.25).

$$y_i = f(x_i) \text{ with } i \in N, N = \{0, 1, 2, 3, \dots\} \tag{2.25}$$

$$\eta = \frac{y_i}{x_i}$$

Output and input are activities, jobs, tasks, utilization, data, or power consumption. Becoming more efficient means doing the same work under the same conditions but reducing the period of time or resources, for example. Energy efficiency is the rate between the consumed energy over time and the useful performed work (performance), shown in Equation (2.26). Measurement unit of the energy efficiency[19] is FLOPS per watt. Lower energy means less power for an activity using the same conditions. In the case of hardware equipment, power efficiency is the ratio between performance and power, see Equation (2.27). It is also known as performance-to-power ratio.

$$energy\ efficiency = \frac{performance}{energy} = \frac{performance}{time * power} \tag{2.26}$$

$$power\ efficiency = \frac{performance}{power} \tag{2.27}$$

### 2.2.6 Graph and Tree Definition

The previous sections described power, energy, performance, and efficiency as aspects for modeling and simulation. In practice, we generalize the system and the diagrams represent an abstraction of the system. Further design process steps are described in Section 2.4.2 and Section 5. A graph or tree represents the system. A graph $G$ consists of a non-empty finite set of vertices $V$ and edges $E$. The finite set of vertices and edges[20] are denoted by $V = V(G) = \{V_0, V_1, \dots, V_n\}$ and $E = E(G) = \{E_1, E_2, \dots, E_m\}$ with $n, m \in N_0$ as any natural number in $N_0 = \{0, 1, 2, 3, \dots\}$. A directed edge[21] connects two vertices[22] and is denoted by a directed pair $(V_0, V_1)$ of vertices with $\{V_o, V_1 \in V, V_o \neq V_1\}$, where $(V_0, V_1) \neq (V_1, V_0)$. Thus, a single directed edge[23] is denoted by $E_1 = (V_0, V_1)$. Finally, a graph is as pair of sets defined as $G = (V, E)$. A tree is a specialized connected graph without any cycles[24] [BR 2012]. The vertical hierarchical levels of a tree are shown in Figure 5.

[19] Energy efficiency: also called energy efficiency ratio
[20] Vertex: node, point, site; edge: line, link, bound
[21] Directed edge: opposite is an unordered pair of vertices $\{V_0, V_1\}$
[22] Vertices: store information and are labeled with $a, b, c$ or $0, 1, 2$ or $V_1, V_2, V_3$
[23] Directed edge: represented as an arrow
[24] No cycles: circuit-free, called acyclic

The index $n$ of a tree level $L_n$ defines the tree depth. In the graphs of Figure 5 and Figure 6, the edge relation stands for the inclusion relation, where an upper level includes many lower levels for all $i \in N_0$. Equation (2.28) shows the level definitions, including the root level $L_0$.

$$L_{i+1} \subseteq L_i \subseteq \cdots \subseteq L_0, \forall i \in N_0 \tag{2.28}$$

The root vertex $V_0$ has the level zero and is the parent of $V_1$ and $V_2$ in this example. A parent is a vertex that is closer to $L_0$ by one edge, or vertex. The vertex $V_1$ also presents a parent of $V_{11}$ and $V_{12}$. Each element $V_{ij} \in V$ can be a root vertex for a new subtree with$j \in N_0$. The children $V_1$ and $V_2$ share the same parent $V_0$ which is formulated as $E_1 = (V_0, V_1)$ and $E_2 = (V_0, V_2)$. The tuples are neighbors because of a connected edge between them. The set of neighbors[25] $Nb$ of a vertex $V_i$ is denoted by $Nb(V) = \{V_i, V_i{}' \in V \mid (V_i, V_i{}') \in E\}$, whereby the numbers of neighbors, equal to the number of edges, in $V_i$ is defined as degree. A vertex with an outdegree of zero is called a leaf: for example, $V_{11}$ in Figure 5 is a leaf. Further information about graph theory and definitions are found in [HHM 2008, Wal 2007].

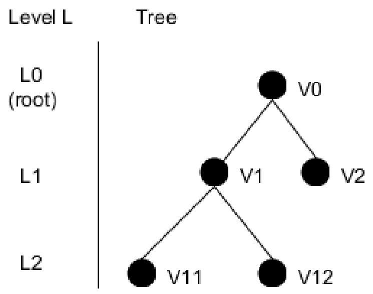


Figure 5: Undirected tree and levels

A tree can be used to define structural aspects of server systems, devices, and states. Figure 6 uses the specialized diagram terminology including the tree elements *type* and *subtype*. The vertices are rectangles, including a label with system information. Figure 6 and the following figures have to be interpreted from top to bottom. The upper rectangle is a *type* at root level and contains different *subtypes,* which are displayed at a lower level. This figure reflects a parent-child[26] relation between *type* and *subtype* whereby $n, m \in N_0$ is any natural number in $N_0 = \{0, 1, 2, 3, \ldots\}$.

[25] Set of neighbors: neighborhood

[26] Parent-child: type (superclass / class) – subtype (subclass)

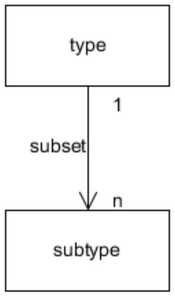


Figure 6: Diagram terminology

*Subtypes* form a subset of *type* in a hierarchical view, because *subtypes* and all their elements are part of *type*. There are defined as: $subtype \subseteq type$ or $type \supseteq subtype$ and represented as $\forall SU_i\{SU_i \in subtype \rightarrow SU_i \in type\}, \forall i \epsilon N$. The set of all members of *subtype* and *type* are defined as follows:

- *Subtype:* $SU = \{SU_1, SU_2, \dots, SU_n\} \quad = \{subtype_1, subtype_2, \dots, subtype_n\}$,
- *Type:* $TY = \{TY_0, TY_1, TY_2, \dots, TY_n\} = \{type_0, type_1, type_2, \dots, type_n\}$, and
- $n \in N_0$ as any natural number in $N_0 = \{0, 1, 2, 3, \dots\}$.

Therefore, the diagram is defined as $\forall SU_i\{SU_i \in SU \rightarrow SU_i \in TY\}, \forall i \epsilon N$. Figure 7 shows a simplified representation of *type* where the root vertex *type* $TY_0$ includes $n$ *subtypes*. The formal definition of the graph $G = (V, E)$ according to graph theory is $V = \{TY_0, SU_1, SU_2, \dots, SU_N\}$ and $E = \{E_1, E_2, \dots, E_m\}$ with $E_1 = (TY_0, SU_1), E_2 = (TY_0, SU_2)$, $\dots, E_m = (TY_0, SU_m)$. The internal rectangles present the subtypes as part of an external higher-level rectangle. Additionally, due to the simplification, the arrows and numbers are not displayed in the subsequent tree figures, because the amounts of *subtypes* are addressed in the number of rectangles in a horizontal manner. The figure becomes more transparent and defined for complex systems, because of a consolidated representation, while not missing any information. A set of all *subtypes* $SU$ represents a smaller scale of depiction.

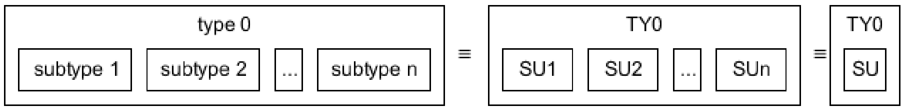


Figure 7: Type and subtypes tree as a diagram

Different vertical hierarchical levels connect *types* and *subtypes*. Consequently, a server system tree is defined as shown in the following equations.

Equation 3: Tree definition

$$\begin{aligned}
\forall s: \quad & l(subtype) = (subtype, type) && => && subtype \subseteq type \\
& l(subtype) = (subtype, \emptyset) && => && subtype = root \\
& l(type) = (type, \emptyset) && => && type = root \\
& l(root) := (root, \emptyset)
\end{aligned}$$

If *type* or *subtype* has no parent, it is the root. The root *type* $TY_0$ is a *subtype* $SU_2$ parent; meanwhile *subtype* $SU_2$ also presents a *sub subtype* $SU_{22}$ parent. The children of $SU_2$ are $\{SU_{21}, SU_{22}, \dots, SU_{2m}\}$ with $m \in N_0$ as any natural number in $N_0 = \{0, 1, 2, 3, \dots\}$. Each element $SU_i$ of $\{SU_1, \dots, SU_n\}$ and $SU_{ij}$ of $\{SU_{11}, \dots, SU_{nm}\}$ are the root vertex for a new subtree and thus follow the same rules of Equation 3.

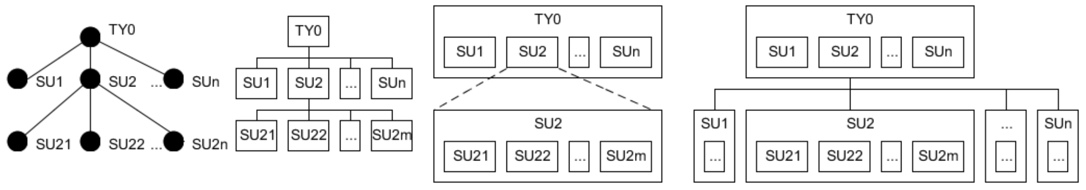


Figure 8: Tree diagram

Figure 9 represents a generic tree including:

- Root *type* $TY_0$ with $TY_0 \in TY$,
- A set *subtype* $SU = \{SU_1, \dots, SU_n\}$, with $SU_i \in SU, TY_0, \forall i \in N_0,\ SU \subseteq TY_0,\ n \in N_0$,
- A set *sub subtype* $SU_1 = \{SU_{11}, SU_{12}, \dots, SU_{1m}\}$, with $SU_1 \in SU, TY_0, SU_{11} \in SU_1 \in SU,\ SU_{1i} \in SU_1, \forall i \in N_0,\ SU_{11} \subseteq SU_1 \subseteq SU,\ m \in N_0$,
- A set *sub subtype* $SU_2 = \{SU_{21}, SU_{22}\ \dots, SU_{2m}\}$, and even *subtype* $SU_n = \{SU_{n1}, SU_{n2}\ \dots, SU_{nm}\}$,

whereas $m, n \in N_0$ are any natural number in $N_0 = \{0, 1, 2, 3, \dots\}$. All *subtypes* $SU$ are given as a set of $\{SU_1, \dots, SU_n\}$. Each member of $SU$ can contain one up to $m$ *sub subtypes* or be an end vertex[27] with no more children. All *sub subtypes* are a set of $\{SU_{11}, SU_{1m}, SU_{21}, SU_{2m}, \dots, SU_{n1}, SU_{nm}\}$[28].

[27] End vertex: leaf

[28] *Sub subtypes* $SU_{nm}$: followed by The Universal Address System for a rooted tree [Che 2015]

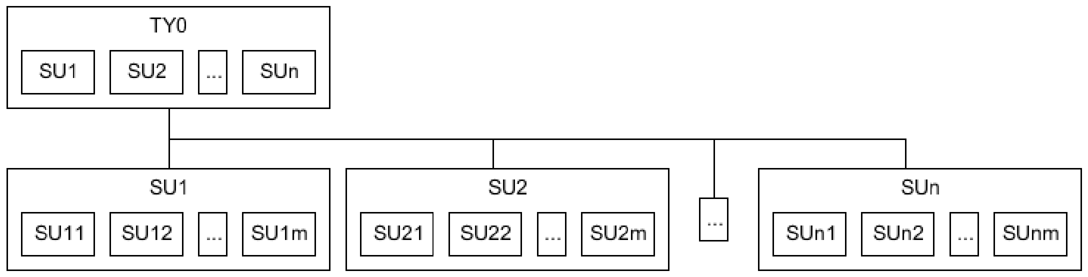


Figure 9: Generic tree diagram

Previously mentioned diagrams and trees define various terms, such as data centers, equipment, environments, servers, systems, and chip domains. This diagram terminology is also valid in the case of state description. In the state diagram, the rectangles are circles or rectangles with rounded corners, shown in Chapter 3.

## 2.3 Data Centers

Data centers (DC) are physical environments, plants, and facilities, which contain and provide Information and Communication Technologies (ICT). Data processing[29] and distribution are the focus within large-scale enterprises. Data centers support all types of applications. Therefore, a pool of data-storage devices, network equipment, information technology infrastructure, and compute nodes are the main hardware resources. Wire cages, power supply, and workload management software (such as load balancer) support the data center functionality.

### 2.3.1 Services

Several data center providers[30] offer computing power or storage services for sale. Specialized Application Service Providers (ASPs) focus on different equipment types. For instance, Amazon Web Services (AWS)[31] offers a service for computing power called Amazon Elastic Compute Cloud (Amazon EC2)[32] . Another service is the Amazon Simple Storage Service (Amazon S3)[33] which can store huge amounts of data. Data is expected to increase to 40000 Exabyte by 2020. Day by day, customers generate and store various data types. The mentioned services are usually cloud services. Cloud services combine different data centers over the Internet, regardless of where they are. Remote access and Internet connectivity enable decentralized environments, such as clouds. Most services are provided on demand and for a certain period only. In general, cloud providers deploy “Anything or Everything as a Service” (XaaS), such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). The infrastructure involves the underlying hardware equipment, such as memory systems and network devices. Amazon Web Services offer Infrastructure as a Service because they provide

[29] Processing: store and manage
[30] Provider: Internet (ISP) -, Application (ASP)-, Full (FSP)-, Wireless Application (WASP) - Service Provider
[31] AWS: http://aws.amazon.com/
[32] EC2: http://aws.amazon.com/ec2/
[33] S3: http://aws.amazon.com/s3/

virtual resources, such as storage. A higher abstraction level is Platform as a Service, which provides an infrastructure and an advanced base runtime environment. Databases or web hosting services are examples where the customer does not care about the operating system. Google Apps[34] is a well-known Platform as a Service provider. The application provides a web-based e-mail program, an integrated calendar, and document creating and processing. Software as a Service provides end-user applications with standard interfaces. It offers access to a holistic environment, such as an application (Gmail, Yahoo correspondence) or social network (Facebook) containing infrastructure and platform parts. Data centers have different requirements due to their use case scenario. Decentralized services support this significant time progress through the virtual combination of various data center resources.

### 2.3.2 Design, Equipment, and Domains

Data centers are split into cabinet rows, including IT equipment, and other all-purpose areas with a multitude of various scopes, such as administration, management, and networking. General-purpose areas of responsibility, also known as key facility systems, are power distribution[35], network (switches, routers), offices (desktop computers), security systems, management, administration, lighting, and *Heating Ventilation Air Conditioning and Refrigeration Technologies* (HVAC, HVACR). Hot and cold aisles containment is established between different rows in well-designed data centers, shown in Figure 10 and Figure 11. It reduces the mixing of hot and cold air, which further reduces the energy demand for electricity required for air conditioning. A row, containing large numbers of 19-inch rack enclosures, has a high equipment density, and a high airflow demand to cool the devices. The equipment density varies between the different facility area types. Consequently, power demands fluctuate in those areas as well as between data centers [TSX et al. 2003].

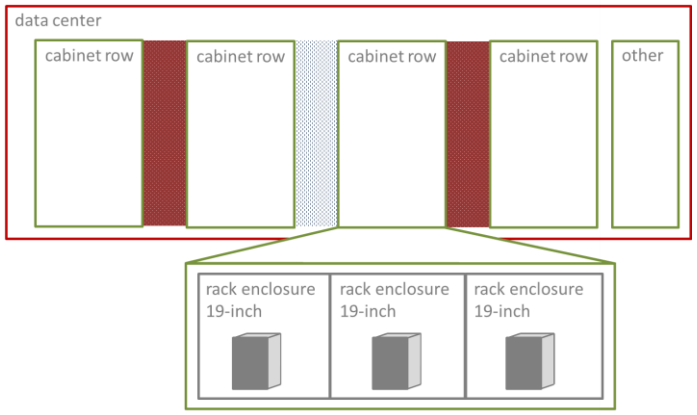


Figure 10: Data center cabinet rows and others

[34] Google Apps: http://learn.googleapps.com/
[35] Power distribution: uninterruptible power supply (UPS) systems

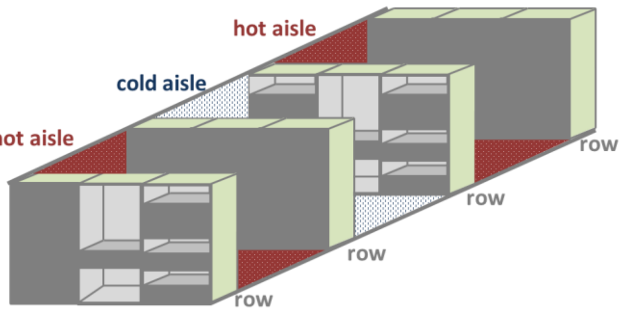


Figure 11: Hot and cold aisle design in data center cabinet rows

A 19-inch rack enclosure is a standardized format for data center equipment. It provides space for mounting many technical equipment modules and resources. Communication systems, backup equipment, and power distributors are peripherals in data centers. A compute node is hardware that is mounted upon the 1U rack unit[36]. One unit is the smallest unit that defines the height of a rack mount. The width is predefined, and comes either from the 19-inch or from the 23-inch rack enclosures. The length varies between 17.7 and 27 inches. Hardware resources and hence the used numbers of resources, their density and temperature development differ because of the various space required by the compute nodes within a rack. Compute nodes are server systems that give computing power for data processing or data storage. We describe further details in Section 2.4.

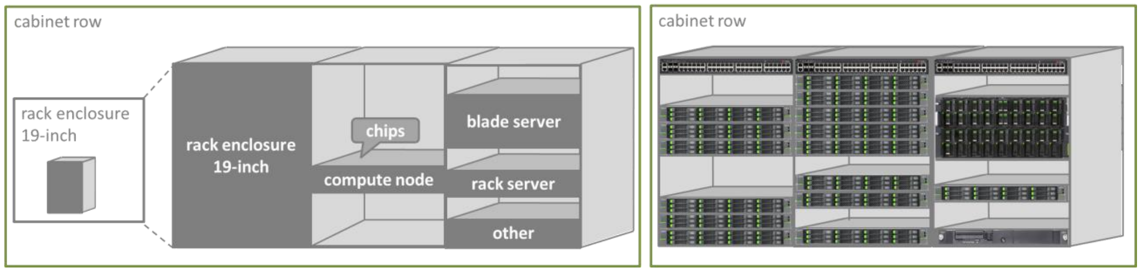


Figure 12: Cabinet row structure

Other enclosures are stand-alone systems, such as desktop, mobile, enterprise, or floor stand[37] computers. Floor stand computers support a higher computational and storage demand in comparison to desktop computers. They are a part of administration, security, and office areas. For illustrating purposes, Figure 13 shows the context between data center equipment. For example, stand-alone enclosures are a subset of other equipment within data centers. However, data centers include cabinet rows and rack-mounted enclosures as well. Memory and storage devices in rack-mounted enclosures are many times larger than in personal computers in order to handle and process the huge amount of data.

[36] 1U: one rack unit, 1.75 inches (4.445 cm) high
[37] Floor stand: also called tower or stand-alone

Network devices for the Internet or intranet connectivity are set up on the Top of Racks (ToR) to provide access to rack-mounted devices and to distribute data. A rack enclosure also mounts storage, input/output (I/O) devices, and power distribution.

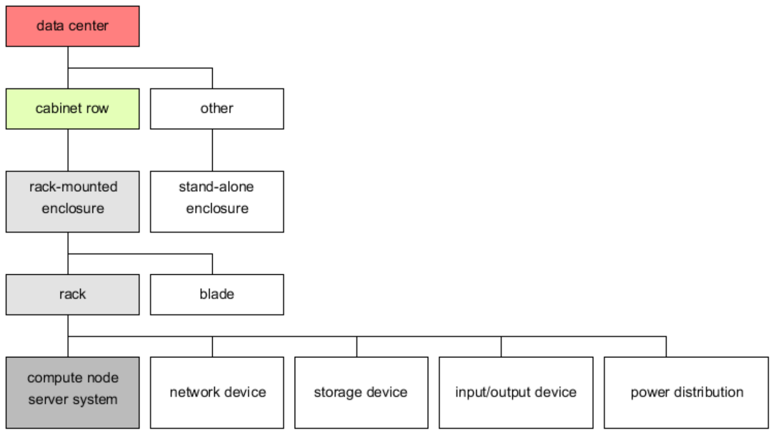


Figure 13: Compute node (server system) in data centers

Rack-mounted compute nodes are server systems that are part of any row within a data center. Rack servers have a higher impact on the DC energy consumption because of their huge number within cabinet rows and the high computing equipment density. The rack server rate within a rack-mounted enclosure is about 40 to 45 percent in all server rooms in German data centers, with 11 up to 100 servers between 2008 and 2010 [HFS 2010]. In contrast, the rack server rate increased to 60 percent in data centers, with up to 5000 servers or larger. Furthermore, rack servers have a market share of about 53 percent, whereas the blade servers, revenue share of total market was 21 percent in the fourth quarter of 2013 [Neb 2014]. Servers with about 80% produce the main energy consumption in data centers. Network or storage devices, both consume the same quantity of energy, was about 10 percent at a data center in 2008 as well as in 2015 [HFS 2010]. Consequently, rack-mounted server systems constitute the focus of this thesis.

The rack server location within a row or data center does not matter, because the thermal and power aspects are abstracted. Incoming tasks, jobs, or services, be it in a virtualized or non-virtualized system, are grouped together as environment conditions. Furthermore, the environment summarizes all external influences and equipment around the rack-mounted server system. Scheduling and placement algorithms are not covered. Additionally, data centers, cabinet rows and rack enclosures are part of the environment domain.

Table 2: Domains, system domain, and examples

| Complexity level | Domain | System domain | Examples |
|---|---|---|---|
| **High** | Data center | **Environment** | Workloads, jobs, queue, tasks, services, rows, hypervisor / virtual machine monitor, lighting equipment, infrastructure equipment |
| | Cabinet row, others, rack enclosure | **Environment** | Network equipment, storage equipment, server |
| | Compute node, server System, component | **System** | Components (processor, memory, bus), software (operating System, firmware, BIOS/UEFI, compiler), architecture (cores, pipelines, caches, switching activities, process, interfaces, protocols), electronic system level (ESL), power supply, connectors |
| **Low** | Chip | **Physical (Chip)** | Circuit, transistor, gate, logic, design, FPGA, ASICs register-transfer, geometry, topology |

The following section describes various server system types, starting at supercomputers and getting progressively smaller to the point of being a server system. Supercomputers are specialized for one application or a small amount of applications in comparison to data centers.

## 2.4 Compute Node Types

### 2.4.1 Supercomputers, Mainframes, and Servers

Specialized IT systems are supercomputers, mainframes, and servers. The Chinese supercomputer called Tianhe-2 is the most powerful supercomputer[38] in the TOP500[39] ranking from June 2014. It has a power demand of approximately 18 megawatts providing a performance of nearly 34 Peta FLOPS, measured by the *Linpack* benchmark. This type of system is specialized for high computing power, usually for a certain application. Particular tasks, such as simulation, modeling or complex computations, are the focus of such systems and include typical application fields such as nanotechnology, human science, or disaster prevention. *OCuLUS*[40] is another high-performance computer at the University of Paderborn.

[38] Supercomputer: high-end computer
[39] TOP500: http://www.top500.org/lists/2014/06/
[40] *OCuLUS*: http://pc2.uni-paderborn.de/hpc-systems-services/available-systems/hpc-cluster/

The data center at the Paderborn Center for Parallel Computing (PC²) provides computing power to its users. The system's theoretical peak performance is about 200 Tera FLOPS. Power information on this system is not available. Nonetheless, power and performance are significant factors of supercomputers.

In comparison to supercomputers, mainframes are smaller systems. Usually, mainframe performance is measured in millions of (machine) instructions per second (MIPS). *Intel* architecture instructions are integer, floating-point, and system instructions. Integer instructions handle arithmetic (ADD, SUB) and logic (AND, OR) operations. The processor's floating-point unit (FPU) executes instructions in either floating-point (real) or integer. System instructions support operating systems via specific commands (MOV). Therefore, MIPS and FLOPS are not comparable to each other. Mainframe servers run many applications and are specialized in data movements, resource processing, and transactions. Mainframes handle huge amounts of input and output data. Enterprise businesses, such as a data warehouses, integrate an Enterprise Information System (EIS), which stores general company data and controls access to them.

The smallest computational nodes within a data center are servers, which support any kinds of applications and operating systems. Servers manage and give access to a network or centralized environment. Depending on the requirements on the server, the server performance is measured using various types of benchmarks. A typical processor benchmark is *SPEC CPU*[41], which compares compute-intensive operations. On the other hand, *SPECpower*[42] evaluates the power versus performance. According to the US Environmental Protection Agency a new server efficiency benchmark was developed, namely. The Server Efficiency Rating Tool (*SERT*)[43] combines power and performance demand over a specific period.

Servers are physical devices in large-scale enterprises, also known as data centers. They have different computing and data processing capabilities because of their various types: the four most well-known types being database, web, image, and application servers [IIZ et al. 2007]. Databases handle the huge amount of user and application data, process and store it. DB2[44] or MySQL[45] are examples of common databases. With web servers, this data is available on the Internet. The third type, the application server focuses on generic purpose software. Application servers support running certain applications and offer a range of services, such as e-learning, sales or search engines. Specialized infrastructure servers distribute the processing load between the various standard server types. The server equipment depends on their use cases and usage models, which are based upon their application types and communication levels. The authors of [DEP et al. 2009] describe the various server types considering the sub-

[41] *SPEC CPU*: Central Processing Unit, http://www.spec.org/benchmarks.html#cpu
[42] *SPECpower*: http://www.spec.org/benchmarks.html#power
[43] *SERT*: http://www.spec.org/sert/
[44] DB2: http://www-01.ibm.com/software/data/db2/
[45] MySQL: http://www.oracle.com/us/products/mysql/overview/index.html

components that have the most impact on server performance. Therefore, different resources and configurations are available. Other examples of server types are gateway, mail, game, and print servers. All of these types can be virtualized and may become equipment in a cloud.

On the other side, chassis types classify servers. Tower servers are not mountable in rack enclosures. They are usually part of small- and medium-scale enterprise facilities. Data centers contain blade and rack servers. A blade chassis is a 19-inch rack-based enclosure, as shown in Figure 14. It provides slots for mounting several devices, offers a high equipment density, and has special features, such as a prewired chassis and shared components like as a power supply unit (PSU), fans, and network devices. The system is ready for plug and play. Blade servers, storage and interface devices are based upon a special slot format. A blade server is a computational node containing a processor, memory, input/output devices, and sometimes storage. In common usage, the term blade or blade server stands for a rack-mounted blade enclosure.

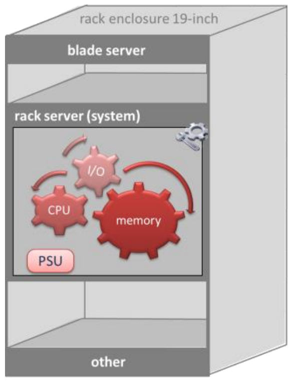


Figure 14: 19-inch rack enclosure

A rack server is comparable with a slot-based blade server. In either case, the regular equipment, such as processors, memory or input/output devices, are part of the physical device. In contrast to blades, a rack unit integrates storage, fans, and power supply units. A compute node is an assortment of various components. Each compute node type contains components that focus on a) high performance, b) safe and reliable operations, or c) low-cost. Consequently, the power consumption of the server differs between these types. Rajamani et al. sum up the server power breakdowns for various compute node types [RLG et al. 2008]. The most frequent reason for server power breakdowns of supercomputers (a) is the power subsystem. Cooling and input/output components cause fewer problems of supercomputer crashes. Mainframe servers (b) are more affected because of insufficient cooling. Standard servers (c) have the most problems with power breakdowns due to their high memory and processor power consumption, as shown in Table 3.

Table 3: Normalized server power breakdown for various classes/types [RLG et al. 2008]

| Compute node type | | Normalized server power breakdown [%] | | | | |
|---|---|---|---|---|---|---|
| | | Power Subsystem | Cooling | Input / Output | Memory | Processor / Cache |
| a) | **Supercomputer, high-end computer** | **35** | 10 | 10 | 15 | 30 |
| b) | **Mainframes** | 30 | **20** | **30** | 5 | 15 |
| c) | **Server (HPC, rack)** | 23 | 7 | 5 | **20** | 45 |
| c) | **Server (blade)** | 23 | 5 | 7 | 10 | **55** |

This short introduction featured various types of computing equipment, from supercomputers to server systems. Because this thesis focuses on server systems with a rack format, the next section will formalize a system and address all major components.

### 2.4.2 Rack Server Systems

This following section defines a rack server system. The specific characteristics are not complete and show the possible parts of a system model. Explicitly used characteristics are described in Section 3 and Section 5. Additionally, related definitions are described in MATLAB notation.

A rack enclosure mounts a compute node, for instance, which is called a system, see (2.29). A rack system is a computational node in a rack format. In general, this is known as a rack server and performs computational work.

```
system       = [{'compute node'}]            (2.29)

rack system  = [{'rack server system'}]      (2.30)
```

In the remainder of this thesis, a system is always a rack server, defined as an overall system. A system $S$ has $one$ up to $n$ parts $S$ with $n \in N_0$ as any natural number $N_0 = \{0,1,2,3,\dots\}$. In general, a system $S$ is defined as a vector $S = \{S_1, \dots, S_n\}$. Equation (2.31) shows the same definition in MATLAB notation[46]. The vector length is equal to the used amount $n$ of parts $S$. Software and hardware are part of the system, as shown in (2.32). In this example, the system has only two major elements.

```
system  = [S1  S2  …  Sn]

        = [{'S1'} {'S2'} … {'Sn'}]           (2.31)

system  = [{'hardware'} {'software'}]        (2.32)
```

[46] MATLAB notation: labels cannot include spaces, using hyphens instead of

A system executes software. An operating system (OS) provides communication between the software and hardware. The standardized application programming interface (API) supports a considerable independence of software and hardware manufacturers. Cloud and virtualization software suppliers benefit from this approach. Various application software types result in complex combinations of computing, network, and storage demand. Processor benchmarks generate different behavior and power consumption in comparison to virtualization benchmarks. This thesis provides realistic application-specific scenarios. Therefore, input data abstracts and defines different application software types. In this thesis, software is defined as $SW = \{SW_1, \dots, SW_n\}$, as shown in (2.33).

```
software   = [SW1 SW2 … SWn]

           = [{'SW1'} {'SW2'} … {'SWn'}]                    (2.33)

software   = [{'operating-system'} {'BIOS-UEFI'}
              {'firmware'} {'application software'}
              {virtualization'}]                            (2.34)
```

Other software parts, shown in (2.34), are server-specific. Firmware is an embedded operating system running on a server-specific baseboard management controller (BMC), which provides management and monitoring capabilities to observe the health and system status. Another motherboard chip or flash device provides the Basic Input Output System (BIOS) and its successor, the Unified Extensible Firmware Interface (UEFI). Both are types of embedded application software.

On the other hand, hardware is any physical device that is mountable in a system. Hardware can have many devices and is defined as $SH = \{SH_1, \dots, SH_n\}$. Hardware distinguishes between three generic types, as shown in (2.36). The size of a rack server defines the mountable and suitable hardware devices for this system.

```
hardware    = [SH1 SH2 … SHn]

            = [{'SH1'} {'SH2'} … {'SHn'}]                   (2.35)

hardware     = [{'component'} {'connector'}
                {'power-supply'}]                           (2.36)
```

A power supply provides electrical power for hardware and electrical circuits (chips). A power supply unit converts incoming alternating current to direct current on different power levels, such as 3.5 or 5 volt. A power supply is defined as $HP = \{HP_1, \dots, HP_n\}$, see (2.37).

```
power-supply  = [HP1 HP2 … HPn]

              = [{'HP1'} {'HP2'} … {'HPn'}]                 (2.37)
```

Connectors are system busses, internal connectors, and external connectors, which are part of a system and connect several components. Busses and caches influence the system performance because of their throughput and latency while processing the data. Busses are not configurable by the consumer and have a constant power consumption of nanowatts or microwatts. Caches are part of the architecture design of a device, such as processor caches or cache lines. The higher the number of cache misses, the smaller the data performance, and the power consumption ultimately grows because of increased repeated requests. All power values lower than watts are considered as static power. Consequently, the single power values of connectors are negligible.

The frontside bus[47] (FSB) connects the processor with the system chipsets, main memory and other peripherals. Other typical data busses are the peripheral component interconnect (PCI), the peripheral component interconnect express (PCIe), the inter-integrated circuit (I2C), the system management bus (SMBus), the power management bus (PMBus), the intelligent platform management bus (IPMB), and the intelligent chassis management bus (ICMB). These internal busses also support internal connectors, such as the front panel and the main power connector. Other external connectors in the case of the front and rear side are serial[48], video[49], and network[50] connectors. Connectors are defined as $HO = \{HO_1, \dots, HO_n\}$, as shown in the following equation:

```
connector   = [HO1 HO2 … HOn]
            = [{'HO1'} {'HO2'} … {'HOn'}]                 (2.38)
```

Connectors and busses are integrated on the motherboard and are not changeable. This setup is fixed after the design phase. The motherboard is part of the component definition. External devices, connected via PCI or PCIe, are not in the focus of this thesis. Part of the system hardware is $SH = \{HC, HO, HP\}$, see Figure 15.

[47] Frontside Bus: equal to processor/memory/system bus
[48] Serial connector: COM / RS232
[49] Video connector: VGA
[50] Network connector: LAN, RJ45

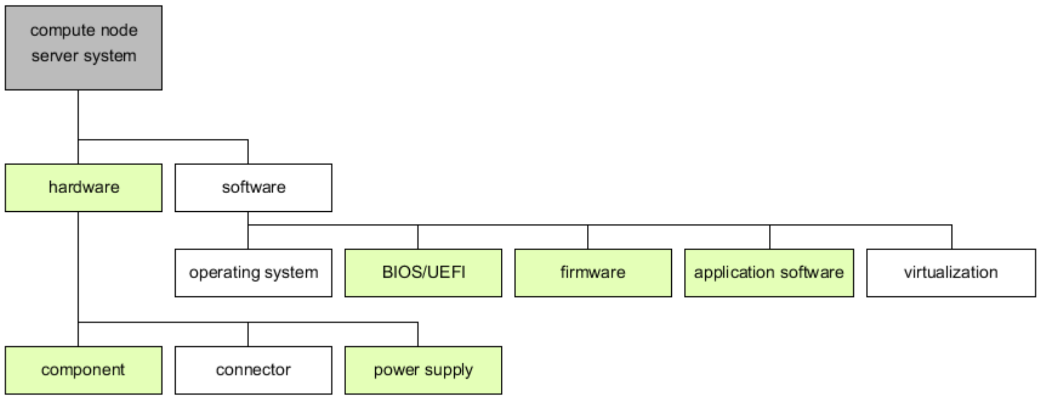


Figure 15: System hardware $SH$ (component $HC$, connector $HO$, power supply $HP$) and software $SW$

The system definition characterizes the used hardware resources and configuration. Software executed on the system is summarized as an application; in contrast, embedded software is abstracted as a configuration. The software vertex of the tree depicts the input parameters of the system model. Further details are described in Section 5.

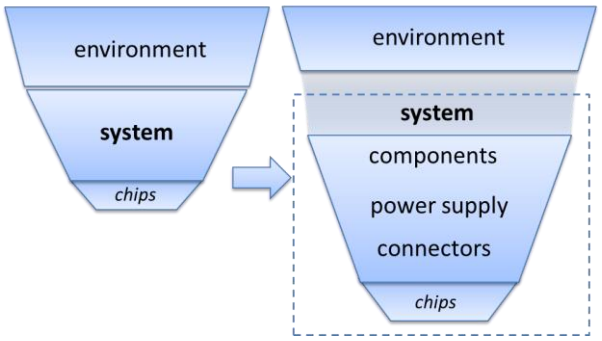


Figure 16: System levels (components, power supply and connectors)

Components $HC$ are real physical parts of the system hardware. The enclosure is part of the hardware but already predefined within our system definitions. Component $HC$ divides n subtypes $HC = \{HC_1, \dots, HC_n\}$, see (2.39). In our example (2.40) components are add-in, onboard, or system-board components.

```
component  = [HC1 HC2 … HCn]

           = [{'HC1'} {'HC2'} … {'HCn'}]                (2.39)

component  = [{'system-board'} {'onboard'} {'add-in'}]  (2.40)
```

Add-in components $CA$ are divided into $CA = \{CA_1, \dots, CA_n\}$. They are specific to the customer, with any amount as long as it is compatible with the server system and connectors. System fans, drive bays (optical drives[51]), local view panel, and expansion cards (network and graphic cards) are examples of add-in components. This differs between server types and is not the focus of this thesis.

```
add-in  = [CA1 CA2 … CAn]

        = [{'CA1'} {'CA2'} … {'CAn'}]                    (2.41)

add-in  = [{'expansion-card'} {'drive-bays'}
           {'system-fan'}]                               (2.42)
```

Onboard components $CO$ are defined as $CO = \{CO_1, \dots, CO_n\}$. These types are not changeable and are provided directly via the hardware, such as the motherboard. Hardware predefines the amount and type of capacitors, transistors, and inductors. Onboard components are all either through-hole devices (THD) or surface-mounted devices (SMD). Temperature and voltage sensors are fixed onboard. Examples of controllers are Ethernet, baseboard management controllers (BMC), or standard north/south bridges. The consumer cannot adapt or change these onboard components.

```
onboard   = [CO1 CO2 … COn]

          = [{'CO1'} {'CO2'} … {'COn'}]                  (2.43)

onboard   = [{'controller'} {'read-only-memory'}
             {'capacitor'} {'transistor'} {'chipset'}
             {'inductor'} {'integrated-circuit'}
             {'regulator'} {'led'} {'sensor'}]           (2.44)
```

In contrast, system-board components can be easily manipulated in a straightforward manner. (Related approaches are described in Section 3.) System-board components $CS$ are defined as $CS = \{CS_1, \dots, CS_m\}$ with $m \in N_0$ as any natural number $N_0 = \{0,1,2,3, \dots\}$, see (2.45). These components are changeable because of their standardized interfaces, connectors, and busses. Otherwise, they are partly predefined by the motherboard. The motherboard supports only special sockets, controllers, or busses. Examples of mandatory system-board components are the central processing unit (CPU), random-access-memory (RAM), input/output devices such as hard disk drives (HDD), etc.

[51] Optical: CD, DVD, or Blu-ray

```
system-board   = [CS1 CS2 … CSm]
               = [{'CS1'} {'CS2'} … {'CSm'}]                (2.45)
system-board   = [{'processor'} {'memory'} {'fan'}
                  {'input-output'} {'others'}]              (2.46)
```

The described component types differ in their variability, power range, and dependencies. The vendor predefines the onboard components, which are neither changeable nor configurable on the system, and consumes the lowest power. On the other hand, add-in and system-board components are individually selectable. They depend on the provided connectors, busses, or slots, which rely on the system-board architecture and generation. System-board and add-in components consume even more power in comparison to onboard components. At system deployment, the processor and memory are mandatory elements during the configuration phase. Consequently, system-board components are the main part of system configuration and deployment. Add-in components are optional elements with a separate order process at any time. Table 4 summarizes the differences between component types.

**Table 4: Comparison component types (system-board, onboard, add-in)**

| Component | Variability | Power [W] | Dependency |
|---|---|---|---|
| **Add-in** | Individually selectable | $10^1 \dots 10^3$ | Connector, slot, port, bus |
| **Onboard** | Predefined by vendor | $10^{-9} \dots 10^1$ | Bus |
| **System-board** | Individually selectable | $10^{-3} \dots 10^3$ | Architecture (system-board), controller, chipset, socket, slot, port, bus |

Some system-board components, such as input/output devices, are optional as well. We divided the system-board components into five major categories, see Equation (2.46). The processor category includes all processing unit devices. The term memory refers to all physical memory device types. The input/output category contains all storage and communication hardware. Internal hard disk drives (HDD), solid-state drives (SSD), or InfiniBand[52] are part of it. Fans are cooling devices that are controlled by temperature algorithms. Parts of other system-board components are optional input devices (keyboard, mouse), expansion cards, and the system motherboard itself. The motherboard provides several essential onboard mounted components, such as BMC. Figure 17 provides a diagram about component types as well as system-board categories, which are further described in Section 5.

[52] InfiniBand: http://www.infinibandta.org/

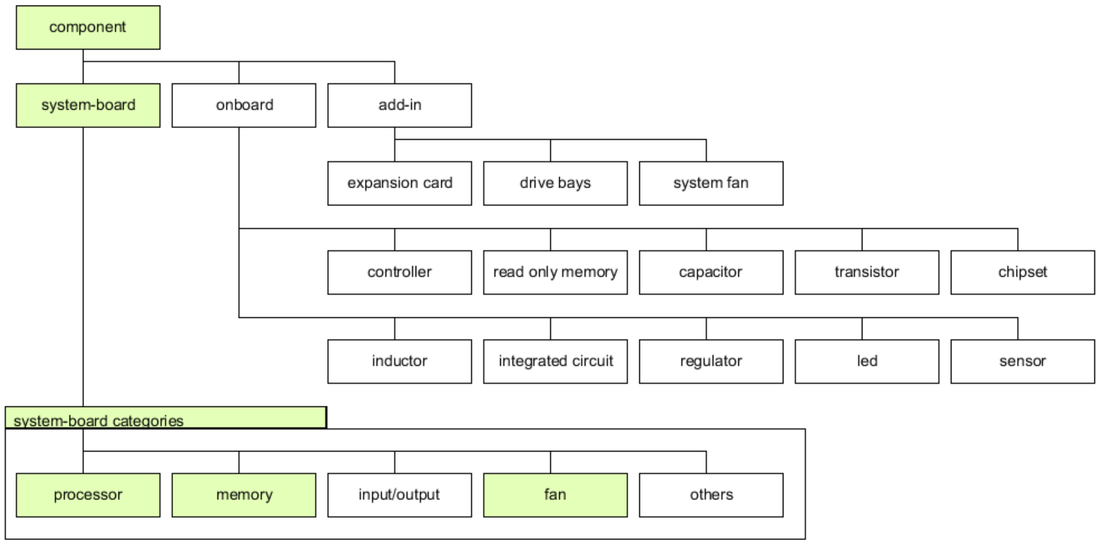


**Figure 17: Component (component $HC$, add-in $CA$, onboard $CO$, system-board $CS$) and system-board categories**

The system definition is applicable and adaptable for blade server, stand-alone servers, or embedded systems as well. For other server types than rack servers, the characteristics of components differ. Many blades within a blade enclosure use, for instance, a shared power supply. Moreover, devices have special form factors compared to rack servers. A blade server does not include the same amount of connectors compared to a rack-based system because of the pre-wired backplane. A blade enclosure can contain more specialized-components because of a higher enclosure size and smaller device formats.

In summary, the hierarchical abstract structure of a rack-mounted server system $S$ is defined by the following components:

- Software $SW$
- Hardware $SH$
    - Components $HC$
        - Add-in $CA$
        - Onboard $CO$
        - System-board $CS$
    - Connectors $HO$
    - Power supply $HP$

Figure 18 shows the abstract server definition as tree $\theta$ with various levels: for example, components $HC$, connectors $HO$, and power supply units $HP$ are part of the system hardware resources $SH$. In contrast, software $SW$ is not divided into detail. Furthermore, components $HC$ include three defined hierarchical component levels: add-in $CA$, onboard $CO$, and system-board components $CS$, which are denoted by $HC = \{CA, CO, CS\}$. The diagram shows an extra level at the bottom, which in turn reflects several subtypes.

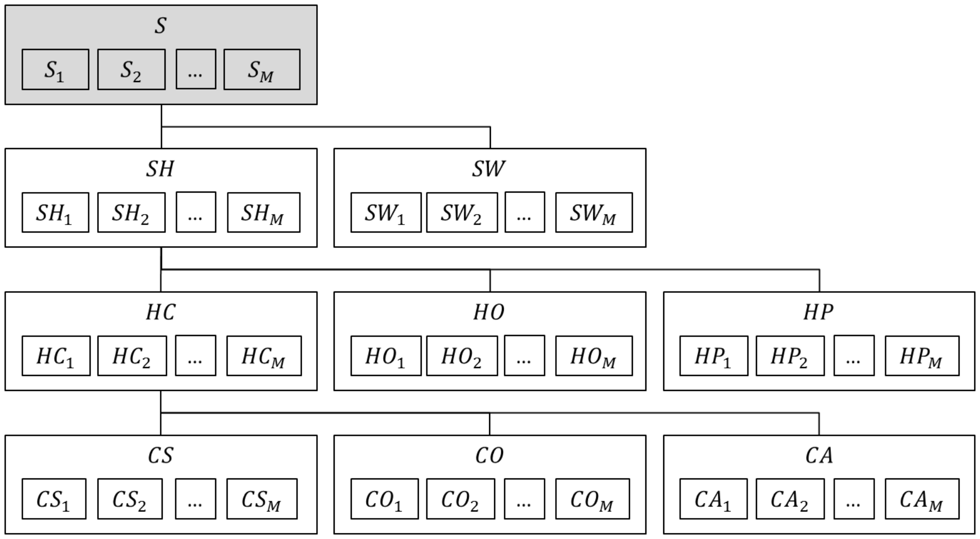


**Figure 18: Hierarchical server system definition**

In this section, we consider the hierarchical structure of the rack-mounted server system in our concept, in which we define the aspect-based component models as part of our server system configuration tree.

## 2.5 Summary

We develop our multi-aspect full-system model by using MATLAB and Simulink. In Section 2.1, we briefly describe MATLAB, Simulink, and its corresponding syntax. In particular, we specify the fundamental terms of this thesis, such as the energy efficiency that we further consider as the performance-to-power ratio. In our evaluation, we use the performance scores of the particular benchmarks that rely upon the utilization levels of the components. The server system is a compute node in a data center, more precisely in a cabinet row that is usually equipped within a rack-mounted enclosure to provide computing power for data processing. The most well-known server types are database, web, image, and application servers in which the power consumption significantly differs. Moreover, we abstractly define our rack-based server system as a hierarchical structure (tree $\theta$) that mainly constitutes software $SW$ and system hardware resources $SH$. We further divide the hardware into the components $HC$, connectors $HO$, and power supply units $HP$. We concentrate on the system-board components, as part of $HC$, which we divide into the following categories: *processor*, *memory*, *input/output*, *fan*, and *others*. The specified configuration tree builds the base of our multi-aspect full-system model.

# 3 Basic Modeling Technologies, Algorithms, and Approaches in Academic Research and Industrial Practice

Peak power consumption and energy efficiency are key economic aspects of server system design and its field of application. High peak power values lead to waste energy and an over-provisioning of server systems. High energy consumption results in a huge thermal increase, especially in data centers, which increases the level of carbon emissions as well as the costs associated with air-conditioning and cooling. Algorithms, such as dynamic voltage frequency scaling (DVFS), influence the power and energy consumption of server systems, yet reduce the performance of the server system at the same time. An aim of this thesis is to model a prototype of a server, including separated single components to analyze the peak power consumption and thermal expansion. This thesis addresses specific power, performance, and thermal models to analyze the energy efficiency of diverse management strategies and server configurations. Energy efficiency aims at minimizing the power consumption by maximizing the performance under thermal constraints. The following sections introduce various technologies, models, and algorithms.

This chapter provides a brief overview of object-oriented modeling techniques. It also outlines some basic modeling techniques and software development processes followed by the intent and purpose of these models. The following section describes various modeling techniques ordered by data, control, and process specification. The model domain influences the model objectives and the stakeholder's point of view. The stakeholder[53] defines the main aspects, priorities, and metrics to verify and confirm the model. Furthermore, we outline the use cases of simulation-based and measurement-based models. The classification of the model depends on the focus at diverse target domains (e.g., power/energy, thermal, and performance) and may change because of power versus cost awareness. The following subsections describe the server system models and simulations. Afterwards, the gap between academic research and theoretical approaches is stated. A table summarizes the used models, considerations, and algorithms. Afterwards, the use case scenarios describe the relevance and significance. They differ with their fields of application. Many companies and customers developed vendor-specific tools for special requirements and functionality. We identify and formulate the aim of this thesis and corresponding problem statement on the basis of proposed techniques, practical realizations, and industrial background.

[53] Stakeholder: user, customer

## 3.1 Modeling Techniques and Domains

Various modeling techniques and software development processes are a part of software engineering and design to ensure high-quality software. Classic software-development processes are the waterfall model and V-model [Lin 2001]. These processes describe the development phases of a system, support design, and system decomposition in the lifecycle stages. Rumbaugh developed an object-oriented modeling technique (OMT) that builds up a system in an analysis, design, and implementation phase [Rum 1991]. The system is transformed into an abstract and structured representation in any manner. The aim of these techniques is to separate between behavior and implementation[54]. Furthermore, the abstract description improves communication among diverse stakeholders. Lifecycle costs decrease because of the stakeholders' better understanding of the design implications, risks, or dependencies [Lon 2012].

Rumbaugh proposes three main types of OMT: the object, dynamic, and functional models, which may be the basis of technical decisions because of system definitions. The characteristic, hierarchy, or usage of a system is defined without any implementation details. An object describes the static structural representation of a system, its architecture and components. Classes model the systemic aspects, including attributes and methods; whereby objects are built according to that definition. Additionally, associations[55] describe the relationships between classes to ensure traceability. The communication among objects, also called interaction, is part of a dynamic model. Furthermore, the model describes the conditions like how an object changes from one state to another. The changeover between various states is a transition, which executes when an external event or action occurs. The functional model includes the flow of information through the system because of the event. These three specification models became standard analysis models in object-oriented software engineering. Using diverse views is an efficient method of presenting complex systems [Tep 2010]. If we completely describe the system, a model becomes an effective prototype [Lon 2012]. Accordingly, the model support designs decisions, changes and also error detections, while checking consistence, correctness, completeness, or relationships. The stakeholder detects problems and trade-offs in early design phase, which reduces errors, development times, and costs [BCC et al. 2014]. The following sections describe the basic approaches to each specification phase.

[54] Concept: separation of concerns
[55] Associations: between classes on model level, links: between objects on instance level

### 3.1.1 Object Specification

The object specification phase aims at analyzing the structure of the system [EG 2000]. A standardized approach[56] is a class and object diagram of the unified modeling language (UML). Data elements, also called objects, show the system in an abstract and variable type by hiding information specifics, which conceal implementation details [Sir 2007]. In such a case, common object classes, sometimes called concepts, generalize a set of data objects. Associations represent the relationship between objects or classes[57]. Subclasses and superclasses generalize this relationship. Figure 19 shows the superclass *component* of a server system, which has three subclasses: *system-board*, *onboard*, and *add-in*.

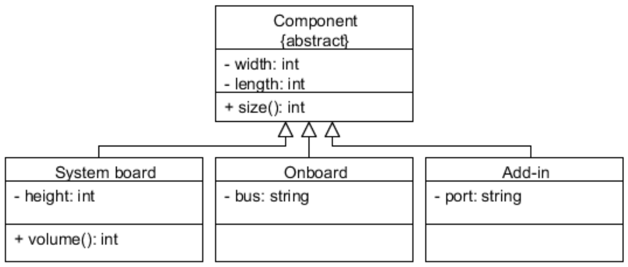


Figure 19: Class diagram – components

The superclass component has two private attributes, *width* and *length,* which define the component floor space[58]. Additionally, the operation *size()* provides a public interface to get floor space information. The three subclasses inherit *width*, *length,* and *size()* but have specialized properties, such as *height*, *bus*, or *port*. Each class can provide various interfaces, features, attributes, or operations, such as calculating the *volume()* for the purpose of area planning. Based on graphical notations, the system is hierarchically in this diagram. The corresponding structure and data aspects of a system depend on the selected diagram type[59]. The data object specification represents the static system; the control specification introduces dynamic system behavior.

[56] Standardized approach: using a modeling language
[57] Associations in diagrams: represented as arrows
[58] Floor space: base area (horizontal space)
[59] Data object diagrams: class, object, package, component, profile, deployment, or subsystem

### 3.1.2 Control Specification

Control specifications define the interaction and communication between objects. In comparison with data object specifications they contain extra information and various control aspects. The dynamic functionality, for instance, behavior or message interchange, is an abstract sequence diagram or other manners[60]. A state or transition approach covers the behavioral perspective as well. A state machine indicates the system behavior[61], a specific system state at any given point in time, whose state changes in response to an external occurring event or action [Oli 2007]. A transition represents the relationship from one state to another [HS 1997]. If there is decomposition of the system into subsystems, it shows the interaction between subsystems and events. A finite state machine (FSM) describes the states and transitions, which is furthermore a deterministic finite automaton (DFA). A 5-tuple $A = (Q, \Sigma, T, q_0, F)$ represents a deterministic finite automaton where the components are as follows [HMU 2001]:

| **Component** | **Description** | **Example (Figure 20)** |
|---|---|---|
| $Q$ | A finite nonempty set of internal states | $Q = \{s_1, s_2\}$ |
| $\Sigma$ | A finite set of input symbols (alphabet) | $\Sigma = \{0,1\}$ |
| $T$ | A state transition function $T: Q \times \Sigma \to Q$ | $t_{ij} \in T$ |
| $q_0$ | An initial (start) state $q_0 \in Q$ | $q_0 = s_1$ |
| $F$ | A set of final (accepted) states $F \subseteq Q$ | $F = \{s_2\}$ |

The previous description provides the example of Figure 20, which shows a state transition diagram (STD). A state transition diagram describes a deterministic finite automaton. States and their changes are major components of this diagram type[62]. A state specifies each object modality in a certain static situation. A transition defines a state change, including various operating conditions. Figure 20 has two states $(s_1, s_2)$ and their corresponding transition $T: Q \times \Sigma \to Q$ referred by input events $\Sigma = \{0,1\}$. The transition functions $t_{ij} \in T$ are a set of triple $(s, i, s')$ with $s, s' \in Q$ and $i \in \Sigma$. The triple consists of the current state $s$, the input event $i$ and the resulting next state $s'$.

[60] Control diagrams: activity, sequence, communication, timing, or state (transition) machine
[61] System behavior: glass box with internal structures
[62] Diagram representation: states represented as circular or rectangular, changes represented as arrows

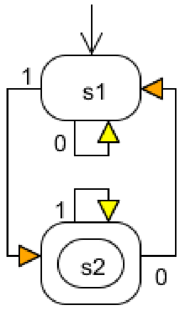


Figure 20: State transition diagram (STD)

In the case of Mealy machine description, each transition function is defined by a current state $Q$ as well as an input symbol $\Sigma$ which returns a state $Q$. A finite set of state transitions labels each state transition $T$. In this example, all transition functions $t_{ij} \in T: Q \times \Sigma \rightarrow Q$ are within a transition table. Table 5 specifies the state changes of Figure 20. A transition $t_{12}$ describes the state change from $s_1$ to $s_2$[63] and is defined as $t_{12} = s_1 \rightarrow s_2$, where a transition $t_{11}$ does not mean any state change. The transition functions are: $T(s_1, 0) = s_1$, $T(s_1, 1) = s_2$, $T(s_2, 0) = s_1$, and $T(s_2, 1) = s_2$.

Table 5: Transition table and transition functions (Figure 20)

| | | Symbols $\Sigma$ | |
|---|---|---|---|
| | $T(Q, \Sigma)$ | 0 | 1 |
| Current State | $s_1$ | $s_1$ | $s_2$ |
| | $s_2$ | $s_1$ | $s_2$ |

| $t_{ij}$ | Set of triple | Transitions |
|---|---|---|
| $t_{11}$ | $(s_1, 0, s_1)$ | $T(s_1, 0) = s_1$ |
| $t_{12}$ | $(s_1, 1, s_2)$ | $T(s_1, 1) = s_2$ |
| $t_{21}$ | $(s_2, 0, s_1)$ | $T(s_2, 0) = s_1$ |
| $t_{22}$ | $(s_2, 1, s_2)$ | $T(s_2, 1) = s_2$ |

Figure 20 shows, for instance, the system states of a server. In practice, a server system state is off $(s_1)$ and a power button event occurs $(i = 1)$. The transition function $T(s_1, 1) = s_2$ defines the system behavior. The server switches to $(s_2)$ only in case of being in $(s_1)$ and when an input $(i = 1)$ occurs. The state of the system changes from $s_1$ to $s_2$. Otherwise, if there is an event zero, the server stays in $s_1$ and nothing changes $T(s_1, 0) = s_1$. As long as no other action occurs, the system stays in the same state. Overall, the state-based approach is the major element of the control specification. The authors of [DFM 2000] give detailed information about formalized state machines, their semantics, and a state transition diagram of a Mealy machine. Control specifications use state machines. Another aspect is how the interaction and communication works.

[63] Transition function: note, the transition $t_{12}$ can also occur in another DFA case if a 0 or 1 occurs

### 3.1.3 Process Specification

Process specifications provide functional models for dynamic system behavior descriptions. They focus on the data and information flow through a system because of a previous event. Furthermore, process specifications indicate the data and information transformation and interaction and also address the information system's functionality. A data flow diagram (DFD) is a graph that shows the flow of data values without any time or control information and its functions that send and transfer data [LD 2000]. It is a top-down process with a gradual refinement, which includes the high-level functionality of a system. Figure 21 illustrates a simple data flow diagram. The *size* of a system determines how much floor space a printed circuit board (PCB) requires. The server *volume* is calculated by taking the *height* into account, whereby server dimensions[64] are predefined at the design process. Both the *size* and the *volume* are required in floor planning tools and the simulation of the computational fluid dynamics (CFD) of data centers to plan the sizing and ventilation[65].

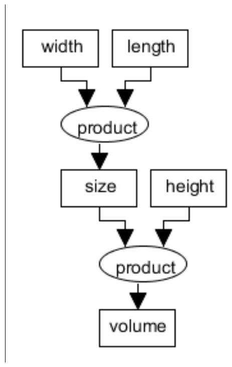


Figure 21: Data flow diagram (DFD) – size and volume data flow

Data flow diagrams are part of the structured analysis/structured design (SA/SD) approach [LPT 1994, Kur 2008]. Additional diagram types[66] support the process specification and define the information flow. The interaction and communication of the system are part of the control specification. Data object specifications define the static representation of the system. Accordingly, the alternative designs and concepts use the model components of all specifications, while ensuring consistency and traceability. Developing similar system models reduces time and costs by re-using model components [BCC et al. 2014, Lon 2012].

[64] Dimension: *height* (1U,2U,4U), *width* (19-inch, 23-inch), *length* (17.7 or 27 inches)
[65] Ventilation: rack- or row-oriented cooling
[66] Process diagrams: flowcharts, pseudo code, entity-relationship diagrams, or data flow diagrams

### 3.1.4 Modeling Domains

State machines, flow diagrams, and block diagrams support system specifications, which are common in a wide range of modeling domains. Figure 22 shows the five major domains defined within the literature [Osi 2010]. Andersson additionally divides the domains, including information, and physical objects and focuses on their major usage or architecture [And 2009, And 2012]. The generic modeling domains are described as follows:

1. The *physical* domain includes objects that describe technical (physical) systems and their designs. The domain includes electrical (power, energy, performance, efficiency), mechanical (motor, rotation), or thermal (heat transfer) systems.
2. Processes, processing structures, activities, operations, architectures, or interactions are part of the *logical* domain.
3. The *conceptual* domain provides the objects, functional descriptions, workflows (control flow), or information of the system.
4. The *contextual* domain focuses on system usages, services, or requirements that are part of the conceptual as well as the external domain.
5. The purposes and constraints are defined in the *external* domain because of predefined vendor limitations, rules, or policies for modeling. Furthermore, the stakeholders influence the model design because of various points of views.

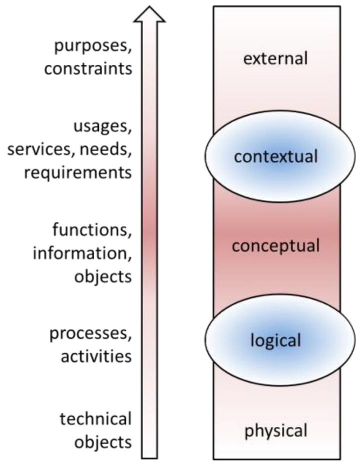


Figure 22: Definition of modeling domains [Osi 2010]

Stakeholders such as the planner, owner, designer, builder, or subcontractor focus on defined modeling domains from top to bottom. For instance, designers focus on the logical system level; in contrast, builders concentrate on the physical level. The defined modeling domains are also part of Zachman's framework that describes the stakeholders and their various perspectives involved in the planning or building phases of the system [PS 2004]. The developers consider various design aspects depending on the role.

## 3.2 Model Objectives, Characteristics, and Criteria

The previous section identified established modeling techniques to deal with various design aspects. The following section describes model objectives, characteristics, and criteria. The overall modeling aim is to build up a physical system. A continuous physical system can be characterized by a model using differential equations $u(t)$. A model $\tilde{u}(t)$ should behave like the represented system $u(t)$ would in the real world. The equations $\tilde{u}(t)$ are an abstraction of the model $u(t)$. Both functions $\tilde{u}(t)$ and $u(t)$ describe system behavior. A model is adequate only under defined criteria. For instance, the area $A$ between $\tilde{u}(t)$ and $u(t)$ are calculated via the integral of the differential function $U(t) = \tilde{u}(t) - u(t)$, shown in Equation (3.1). The real and represented systems are identical when area $A$ is zero, which is not the case in practice. Thus, the customer specifies the criterion $a$, which decides when a model is adequate or not. The model is suitable if the area $A$ is lower than the criterion $a$.

$$A = |U(t)| = \int_{-\infty}^{\infty} |U(t)| dt = \int_{-\infty}^{\infty} \left|\left(\tilde{u}(t) - u(t)\right)\right| dt$$

$$A = \begin{cases} < a, & adequate \\ \geq a, & not\ adequate \end{cases} \tag{3.1}$$

The represented physical system *u* is defined as a time-continuous system and contains time-continuous[67] values $x(t)$. In practice, the input signal $x(t)$ is a continuous value series over continuous time. An alternative description of the system is time-discrete[68] $x[n]$, with *n* as any natural number $n \in N_0, N_0 = \{0, 1, 2, 3, \dots\}$. A sample identifies a value $x(t)$ at a certain point in time $t_N$ (or simple $n$) using an interval $T_N$ as the interval between $x[n]$ and $x[n+1]$. Therefore, a time-continuous system becomes time-discrete. A discrete system is represented as a finite or countable infinite set of values, which is as a sequence $x[n * T_N] = \{x[0], x[T_N], x[2T_N], \dots, x[nT_N]\}$. The right side of Figure 23 shows a time-discrete system $x[n]$ with $n = \{0,1,2, \dots, 9,10\}$, and $T_N = 1$ sampled from the left side of the figure represented as $x(t)$. Both diagrams show the same period of time $0 < i \leq T(n * T_N)$.

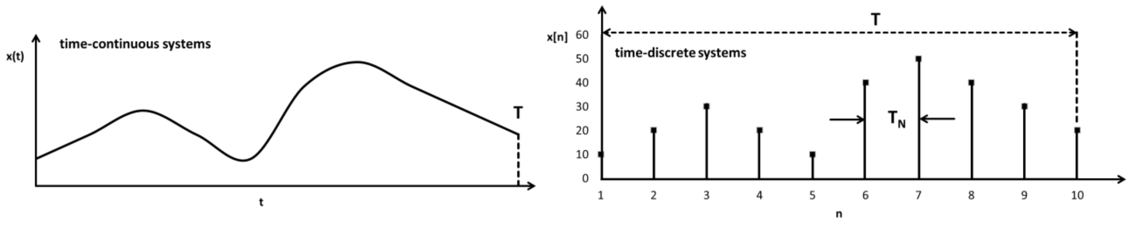


Figure 23: Time-continuous $x(t)$ and time-discrete $x[n]$ systems

[67] Time-continuous: differential equation specified systems (DESS)
[68] Time-discrete: discrete-time specified systems (DTSS)

Time-continuous and time-discrete systems are mappable into each other, as shown in Figure 24 [Gee 2004]. The sampling theorem by Nyquist-Shannon, assuming that the signal is represented in the frequency domain, defines the following frequency dependency to avoid information losses:

> *"The sampling frequency should be at least twice the highest frequency contained in the signal."* [Ols 2000]

The corresponding mathematical equations are shown in Equations (3.2) and (3.3) with the sampling frequency $f_N$ and the highest frequency $f$ in the system. The sample rate $T_N$, or frequency $f_N$, ensures the reconstruction from time-discrete to time-continuous systems.

$$f_N = \frac{1}{T_N}[Hz] \qquad f = \frac{1}{T}[Hz] \tag{3.2}$$

$$f_N \geq 2 * f \tag{3.3}$$

For instance, temperature sensors usually have a sample time $T_N$ of approximately one second. A smaller sample time offers no advantage because on the one side, the traffic[69] limits the performance of the bus connection; and on the other side, the fan's inertia ensures that the fan speed is controlled within a defined time response from generally ten seconds[70]. Accordingly, the additional values, because of a shorter sample time, are practically useless and provide no further information. A correct sample time immediately captures all temperature changes. In comparison, a longer sample time may lead to wrong or missing values. If the sample time is insufficient, multiple temperature changes occur within one sample time. The various changes and their effects are not possible to capture. In the worst-case, we do not find any temperature change. Accordingly, the monitored time and the sample time should fulfill the requirements as stated before. In contrast to thermal sample times, the sample time of power values can vary between micro- and milliseconds, depending on their domain. For instance, the changes to the input signal and their effects are not determinable in the case of the complete system's inertia. An increasing processor frequency and voltage scaling produces a temperature change within seconds. Because of the higher temperature, the fans will wind up and consume more power in comparison to the previous state. Both the temperature and the power consumption grow slowly and are subsequently monitored in comparison to the dynamic voltage and frequency scaling. In the case of measurements, the continuous time is sampled to time-discrete series to avoid a huge mass of values.

[69] Traffic: requests to other sensors and therefore, delays in response

[70] Time response: fan vendor specific, less than 10s leads to acoustic and audible roar from the fans

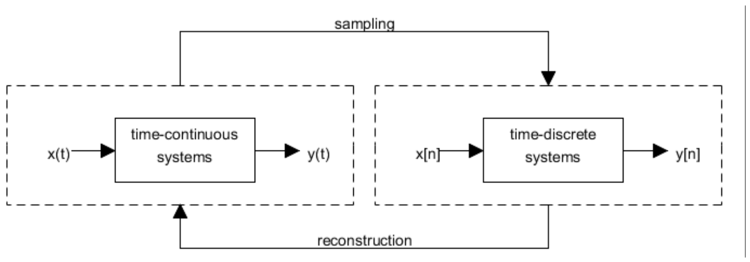


Figure 24: Time-continuous and time-discrete cycle

Parameter changes influence the system behavior. For instance, the power consumption of a memory module depends on the voltage and frequency settings and therefore, we utilize it in another manner. The power consumption *p* of a component depends on the utilization level[71] at a specific time. In this case, the utilization level is defined as $u(t)$. The system generates the power consumption $p(t)$ as an output signal. A short notation is known as $u(t) \rightarrow p(t)$. The calculation of energy consumption $E$ requires the time period $0 \leq t \leq T$. Equation (3.4) defines the summation of power values in a time-varying (LTV) form. According to the time period, an integral sums up the related values $p(t)$.

$$E = \int_{t=0}^{T} p(t)\, dt \tag{3.4}$$

On the other hand, the input values of the system, the utilization levels $u[n]$, and the resulting output power value $p[n]$ can be part of a time-discrete system. A short notation is known as $u[n] \rightarrow p[n]$. In a time invariant (TIV) system, the output values result in a time-varying convolution, as shown in Equation (3.5). The time period limits this summation of discrete-time values $p[i]$. It simply adds up a set of values defined by each point in time $i$, with $i$ as any natural number.

$$E = \sum_{i=0}^{T} p[i] \tag{3.5}$$

Consequently, it results in four combinations of time and value characteristics, shown in Table 6. In use case (*I*), the signal and time can take on any values as already described in Figure 23. The discrete times are a sequence $T = \{t_0, t_1, t_2, \dots, t_n\}$ and analogous to this, the discrete values are $X = \{x_0, x_1, x_2, \dots, x_n\}$. Both in combination result in use case (*IV*), whose values and times are discrete, as shown in Figure 25. At the discrete time $(t_3)$ only one discrete value $(x_2)$ or $(x_3)$ exists. A value $(x_{2.5})$ is invalid and does not exist because it is not part of the sequence *X*. Use case (*III*) shows that the sample time $T_N$, the interval between $x[n]$ and $x[n+1]$, varies because of discrete values. In this example, the sample time $T_N(x_3, x_4)$ is much smaller in comparison to $T_N(x_4, x_3)$. Figure 25 shows all four signal sampling and discretization use cases.

[71] Memory utilization level: voltage and frequency dependencies not covered, implicit via utilization

Table 6: Use cases for continuous and discrete time and values

| Use case | Time characteristic | | Value characteristic | |
|---|---|---|---|---|
| *(I)* | Continuous | $t$ | Continuous | $x(t)$ |
| *(II)* | Discrete | $t_n$<br>$n$ | Continuous | $x(t_n)$<br>$x[n]$ |
| *(III)* | Continuous | $t$ | Discrete | $x_n(t)$ |
| *(IV)* | Discrete | $t_n$ | Discrete | $x_n(t_n)$ |

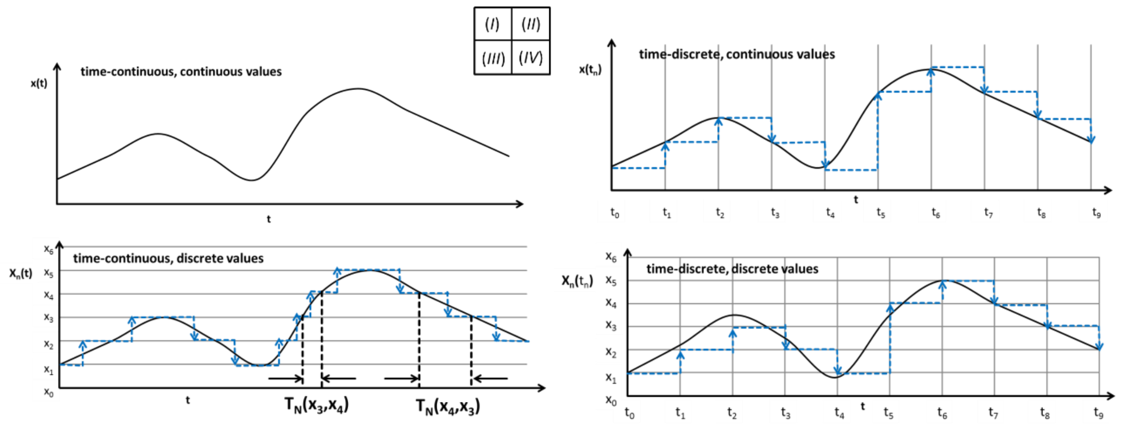


Figure 25: Signal sampling and discretization

The power consumption $x(t)$ varies over a period of time, which is a time-varying process. The value $x(t)$ depends on a specific point in time $t$. The time-varying behavior becomes time-independent when the time is no longer considered. For instance, the largest value of the signal is time-independent because only the global maximum $(x_5)$ is relevant, independent of how many local maxima $(x_3)$ and $(x_5)$ occur at which time. A local maximum is calculated by the first derivative, which is set to zero. The resulting critical points of $f$, values $x_0$, are used in the second derivative to check the extremum type. The function $f(x)$ has a local maximum in a given interval[72] $I$ when the second derivative in $x_0$ is negative, the value $x_0$ is part of the defined interval, and the function values $f(x)$ are smaller than the value of $f(x_0)$ for all $x \in I$, see (3.6). Furthermore, it is a global maximum, whereas consistently all other values $f(x)$ are smaller than $f(x_0)$ within the complete function. The peak power value $(x_5)$ is used to choose a suitable power supply unit for the largest power consumption of the system to avoid over-provisioning.

$$\frac{df}{dx} = f'(x_0) = 0, \ \frac{d}{dx}\left(\frac{df}{dx}\right) = f''(x_0) < 0 \tag{3.6}$$

$$f(x_0) \geq f(x) \tag{3.7}$$

[72] Interval: if an interval is not defined, it is described as values x ‘near’ $x_0$ or nearby

A model depicts the present state of a system at a specific time. Usually, this state has a present value of a characteristic set. The parameters differ in model characteristics and criteria. The power model of the rack-mounted server system can estimate the main component-based consumer on the basis of power states or being more granular on the basis of instructions. In the case of instructions-based analysis, the calculation time increases because of a more complex model than a model with fewer details, more parameters, and a higher sample rate. A rudimentary level of detail cannot guarantee a high-granular analysis of power values. For example, a component has a power value *p* at a specific point in time. A discrete-time interval is defined as $t \in [0, T]$ with constant time steps $\Delta t = 1$ and a time period of $T = 3$. A calculation[73] of energy values $E = \int_0^T p(t)\,dt$ becomes a) more complex if the time interval expands like $T = 5$ or b) more exact if intermediate steps are added like $\Delta t = 0.5$. On this basis, the calculation time will increase. Therefore, it is necessary to define the relevant accuracy of the model and the corresponding aim [Paw 1990]. The deviation between estimated and measured values decides if a strategy is applicable. Another model aim, added to build up a real system, is to design a simple model that uses the minimal input parameters and reduces the model complexity. Simplicity and accuracy are contradictory. A more exact model requires a more complicated model and usually the sampling rate increases to ensure state changes. A complex model generates overhead and consumes more computational time, which is not applicable for real-time approaches. The model level and level of detail[74] are results of the model accuracy decision of stakeholders [SMA et al. 2003]. Depending on the model's intentions, various modeling levels may be more suitable than others might be. The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) specifies the product quality model which describes quality characteristics and criteria, as shown in Table 7 [ISO 2011]. Each top-level characteristic (factor) has many sub-characteristics (criteria) which are analyzed by metrics, such as lines of code (LOC) or special performance metrics like jitter, latency, response time, or throughput. For instance, the accuracy criterion is assessable by the amount of internal states and by the supported input parameters. The product quality criteria and related metrics are customer-specific [CHW et al. 2010].

[73] Calculation: also known as estimation or prediction

[74] Level of detail: granularity

Table 7: Product quality model and characteristics by ISO/IEC 25010:2011 [ISO 2011]

| Characteristics (factor) | Sub-characteristics (criteria) |
|---|---|
| **Functionality** | Suitability, accuracy, interoperability, compliance, security |
| **Reliability** | Maturity, fault tolerance, recoverability |
| **(Re-) Usability** | Understandability, learnability, operability, attractiveness |
| **Efficiency** | Time behavior, resource utilization |
| **Maintainability** | Analyzability, changeability, stability, testability |
| **Portability** | Adaptability, installability, co-existence, replaceability |

Moreover, a model should be generic as part of the usability or portability characteristic. For instance, one metric is the variety of supported rack-mounted enclosures, such as rack or blade servers, as well as stand-alone systems, like desktop or mobile computers. In this thesis, we use the model that deals with various classes of systems and is adaptable by adding new classes. More details are described in Chapter 4.

A physical system modeling can be based upon measurements using continuous or discrete values. The system behavior is based upon simulation results and is described in the next sections. Afterwards, a short overview about configuration-based and optimization-based models is given.

### 3.2.1 Measurement-based Models

The modeling properties of parameters have diverse impacts, caused by the chosen model style. A standard practice is to model and depict current system behavior via sampling or discretization. The numeric parameter type, such as floating-point instead of integer, influences the time resolution and for that reason, influences the calculation time. If a system's behavior and values are characterized by data and the findings of a real-life system and its respective hardware, it is a measurement-based approach. Potential areas of interest are observed. The customer analytically characterizes the measurements of the system in order to find dependencies between associated instructions or functions. Therefore, detailed information about the real hardware, such as its architecture, is beneficial. Mathematical analysis techniques form the basis of the model. In the case of power or energy models, stress tests utilize the system components to check the component power (states) and corresponding performance characteristics such as the computation time. Benchmarks use synthetic workloads to test real-world systems or discrete system components in a specified and reproducible manner under defined circumstances. Many experimental measurements and iterations[75] support the system evaluation, verification, and confirmation as part of the development process.

[75] Iterations: choosing various parameters

Especially the power measuring and profiling are key technologies for creating a sufficient model. The measurements are hardware-based when additional meters, integrated sensors, or special instruments observe the current and voltages of the hardware devices. In comparison, the software-based measurements' aggregate power values from the operating system or application, such as hardware counters, which are flexible in comparison to the hardware-based method.

### 3.2.2 Simulation-based Models

Simulation-based models forecast the system behavior under not yet tested or measured circumstances. The models analyze and abstract the system behavior, but the internal architecture, operation, or configuration is unknown. A vendor-specific component is a black-box[76] device and only names or interfaces are visible. Another benefit is that the physical device is not required. Sometimes simulation-based models are called predictive or execution models. Comprehensive approaches based on system-theoretical models are defined that include analysis results of previous system generations and their behavior. The model is partial because some characteristics are not predictable. We model only a limited number of variations because of time constraints (execution time, time to market). Complex systems often use simulation-based approaches [SMA et al. 2003]. Simulations use either a behavior description or an algorithm level for the functional behavior of hardware components. Certainly, the power characteristics of tasks are estimated by table-based approaches, including activities, transactions, or instructions. The power consumption of the overall system is based upon simulation results. Statistical analysis and stochastic methods are basic elements of an algorithm level [LK 2000, AK 2002]. A simulation-based approach predicts a future configuration, accesses, transactions, or activities. Running an empirical simulation supports all levels of existing and non-existing functions, algorithms, or physical devices. Experimental results are especially extrapolated about non-existing system conditions. On the other hand, empirical results are not available or possible. Either the hardware of a future component does not exist, or it may be either too dangerous or costly to test [FOG 2008]. Another aspect is fault injection in which various fault conditions are simulated and analyzed[77] [EL 2009, Hex 2003, SSH 2014].

### 3.2.3 Configuration- and Optimization-based Models

Another modeling style reflects the experimental environment. A configuration model describes physical observations in relation to random settings of characteristic parameters. A randomly generated sequence has properties, ranges, or limits to answering a specific question. Usually, the aim is to minimize or maximize certain model output parameters, functions, or values under defined constraints. In complex systems, a parameter influences the results in a positive and negative direction at the same time as cost and time. So, solving the

[76] Black-box: outside point of view
[77] Analysis: fault tree analysis (FTA)

problem has many conflicting objectives[78], which are defined as vector *F,* including multiple, potentially, conflicting objective functions $f_1, \dots, f_k$. The objective is to minimize or maximize all of them simultaneously at the same point in time, where *k* defines the maximum number of objective functions, with *k* and *n* as any natural number [SWK 2011, DSH 2005], as shown in following equations.

$$\min_{x \in \mathbb{R}^n}\{F(x)\} \text{ or resp. } \max_{x \in \mathbb{R}^n}\{F(x)\} \quad (3.8)$$

$$F(x) = (f_1(x), f_2(x), \dots, f_k(x)) \quad (3.9)$$

$$F: \mathbb{R}^n \to \mathbb{R}^k,\ k, n \in N_0, N_0 = \{0,1,2,3,\dots\},\ k \geq 2 \quad (3.10)$$

A single solution does not exist, and the set of correct trade-off solutions called Pareto[79] optimal (PO) sets are defined. These sets of optimal solutions are mathematical equally correct solutions within the multi-objective optimization (MO or MOO) approach. Additionally, constraint functions *C*[80] characterize the subject functions as follows [Mos 2005]:

$$C(x) = (c_1(x), c_2(x), \dots, c_m(x)) \leq 0 \text{ or } C(x) \geq 0 \quad (3.11)$$

$$i \in N_0, N_0 = \{0,1,2,3,\dots\}, \quad (3.12)$$

where the vector of decision variables is defined as:

$$X = (x_1, x_2, \dots, x_n)^T \quad (3.13)$$

The decision vectors are part of a workable region, which is a set of constraints *C*. These conditions should be satisfied among the set of all vectors, which is a set of favorable (e.g., non-dominated) solutions in the objective space also called Pareto optimal front or Pareto front. Various multi-objective algorithms are described in [Abr 2005, BDM et al. 2008, Deb 2002, ES 2003, GC 2000].

Measurement-based and simulation-based approaches form the primary basis for building up real-world systems. Depending on the model's intentions, various modeling tiers are more sufficient than others. Especially the model domain plays an important role with regard to functionality characteristics and design decisions.

## 3.3 Server System Model Domains and Aspects

This thesis focuses on rack-mounted server system models and their simulation. As proposed before, a model description relates to corresponding objectives. In the case of servers, the maximal outlet, also known as the exhaust temperature and the related airflow are critical characteristics in the planning phase of data centers. High costs of data centers incur from

[78] Multiple conflicting objectives: known as the multi-objective optimization problem (MOP)
[79] Pareto: Vilfredo Pareto generalized the concept of Francis Y. Edgeworth
[80] Constraints: define the variable boundaries (low or high) of each decision variable $x \in X$

cooling the equipment, which results in energy costs. Figure 11 shows the hot and cold aisle design in data centers. As a consequence, each consumed electrical power unit results equal quantity of released heat [ERK 2006]. Air-conditioning systems[81] dissipate heat and coldness to keep the data center temperature and humidity conditions constant. High humidity supports condensation on electronic devices, which may lead to electrical short circuits. In addition, critical server components could be damaged, and a server crash and shutdown may result from these damages. Servers produce masses of heat. Due to an increasing power density[82], the heat generation grows exponentially. The authors in [Sku 2013] assume approximately 30 kW in power loss in a 19-inch rack enclosure. Thus, heat removal becomes a fundamental issue for data centers. Data centers use entire systems, such as HVAC (Heating, ventilation, and air conditioning) equipment, to ensure permanent heat dissipation. A consequent heat production exists in all domains up to chip level, as shown in Table 8. The demand of heat dissipation in entire data centers is hundreds or thousands of times larger in comparison to the chip level.

Table 8: Thermal systems in various domains

| Thermal demand | Domain | Thermal systems |
|---|---|---|
| **High** | Data center | Heating, ventilation, and air conditioning (HVAC), heating, ventilation, air conditioning, and refrigeration technology (HVACR), computer room air conditioner (CRAC), computer room air handler (CRAH) |
| | Rack enclosure | Air distribution (fan panel), water cooling, active flow control (AFC) |
| | Server system, Component | Air distribution, ventilation (fan), water cooling |
| **Low** | Chip | Heat spreader, heat sink |

The other remaining costs are electrical equipment costs. These energy costs are associated with operating and maintaining (O&M) the IT infrastructure or computing equipment, which are operational expenditures (Opex). Opex are ongoing, regular occurring costs in operation, such as administrative, lightning, or thermal costs as well as power expenses over a given period, e.g., a year. In contrast, capital expenditures (Capex) are fixed one-off costs[83], for instance, initial IT infrastructure and equipment investment costs, to ensure the daily business and services of data centers. All cost types are summed up into the total cost of ownership (TCO) and are a critical part of return on investment (ROI) decisions, which faces the trade-off

[81] Air-conditioning systems: filtering, reheating, humidifying, and dehumidifying
[82] Power density: effect of shrinking CMOS sizes, watts per square cm
[83] One-off costs: cost to build

between buy[84] and internal build [HLH et al. 2012]. There should be a balance between TCO, ROI, and power usage effectiveness[85] (PUE). This metric, defined by *The Green Grid*, compares total data center energy consumption with the IT equipment energy consumption. A PUE value results by comparing the complete utility load with the total IT equipment load [Mat 2011, Jau 2011]. On the other hand, uninterruptible power supply (UPS)[86] systems have to be able to handle all energy requirements within the high-availability 24x7 data center if a power failure occurs. In decentralized or hierarchical data centers, power distribution units (PDUs) give the necessary power to many servers under normal, everyday circumstances. PSUs distribute power to single servers and have a smaller power range in comparison to PDUs. Table 9 shows the classification of the power systems to their domains. For instance, power models at data centers handle power demands in kilo-, mega-, or gigawatt. Chip's power model calculates demands as microwatt and milliwatt. A data center operator monitors the power and energy values in an entire building [New 2008].

**Table 9: Power and energy systems in various domains**

| Power (energy) demand | | Domain | Power systems |
|---|---|---|---|
| | **Power [W]** | | **Examples** |
| **High** | $10^6$-$\infty$ | Data center | Uninterruptible power supply (UPS), energy power supply (EPS) |
| | $10^4$-$10^{5-6}$ | Rack enclosure | Power distribution unit (PDU), rack distribution unit (RDU) |
| | $10^2$-$10^3$ | Server system, Component | Power supply unit (PSU) |
| **Low** | $10^{-3}$-$10^1$ | Chip | DC-DC power converter, control unit, transformer |

The data center manager should be aware of the growing possibilities to use information, which leads to adjustable data center capacity, such as computational nodes and storage. A data center manager has to plan the energy demand for peaks. The power systems should be able to handle extra energy demands also within the near future. Thus, the authors of [RN 2011] introduce a growth model to discuss future IT power requirements. The growth model includes the following two main parameters: a) the design IT load profiles and b) the system capacity plan. The profiles predict the actual, initial, minimum, and maximum (final) load in the data center. The system capacity plan supports the defined IT load profiles via step size and margin. If the system capacity is bigger than the actual IT load, capacity, energy, or costs are wasted. However, if the system capacity is lower in comparison to the real IT load, the data center needs extra IT resources to avoid crashes.

[84] Buy: collocation, hosting
[85] PUE: reciprocal data center infrastructure efficiency (DCiE), aim: low PUE value
[86] UPS: have batteries, converters, and generators

Servers in data centers are utilized in various ways. Various workloads require diverse server power and performance levels. On the one side, some services and operating times vary in relation to the time of year and day: for instance, a peak utilization level of a mail server will occur when the working time in an office begins. Customers check and retrieve emails from the mail server in a concentrated manner. After the first working phase, the requests (i.e., utilization level) decrease. Another example is the use of a productive subversion (SVN) server. Figure 26 and Figure 27 show the total number of commits within one week and within one day over a period of one year. The statistics identify differences at the working days and hours. Monday and Wednesday are used more often to commit changes than on Tuesday, Thursday, and Friday. Fewer commits occurred over the weekend. On the other side, the numbers of commits increase during working hours. Peaks are displayed at 9am, 2pm, and 5pm. Each commit creates input and output traffic at a subversion server. Processors, memories, networks, and storage systems handle and process this traffic. Therefore, the server system and component utilization vary.

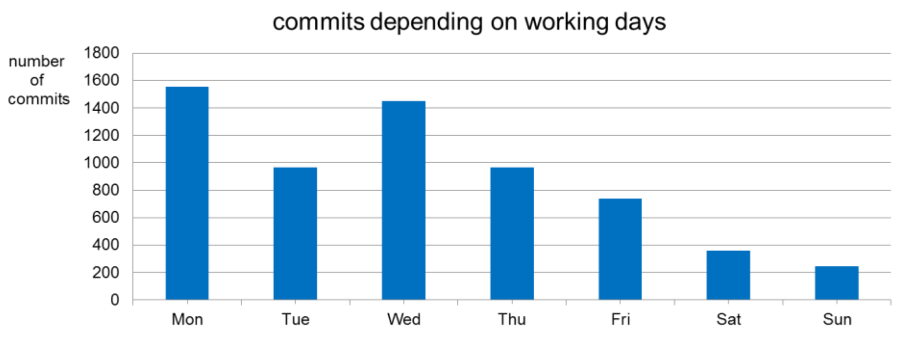


Figure 26: SVN server commits – day statistic

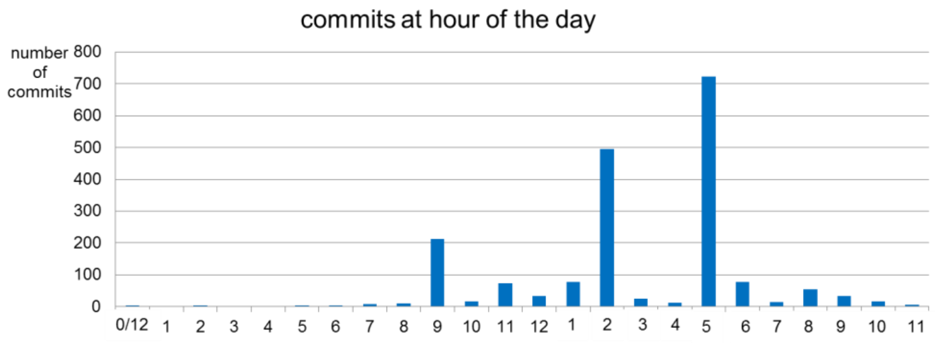


Figure 27: SVN server commits – hour statistic

Previous studies discovered that server utilization could be extremely low, because of long idle times between jobs. Figure 27 shows this effect at early-morning hours. The commits are done marginally in comparison to office hours. The vendor does not design the server for an average utilization level of approximately 10 up to 30 percent. Invested capital costs and resources are wasted because the server is underutilized and consumes high power in low utilization (less energy efficiency[87]) in comparison to a high utilization level with a high energy efficiency level, as shown in Table 10 and Table 11. High energy costs result at low utilization without any processing benefits. Both tables show less energy efficiency at utilization levels from 10 up to 30 percent, which increases by higher utilization. In the case of optimized systems ($II$) the initial power is only 10 percent instead of 50 percent of peak power [BH 2007].

**Table 10: Server utilization, power, and energy efficiency – system $I$ [BH 2007]**

| Utilization [%] | Power [% of peak] | Energy efficiency |
|---|---|---|
| 0 | 50 | 0 |
| **10** | **55** | **0.18** |
| **20** | **60** | **0.3** |
| **30** | **65** | **0.46** |
| 50 | 75 | 0.66 |
| 70 | 85 | 0.82 |
| 100 | 100 | 1 |

**Table 11: Server utilization, power, and energy efficiency – system $II$ [BH 2007]**

| Utilization [%] | Power [% of peak] | Energy efficiency |
|---|---|---|
| 0 | 10 | 0 |
| **10** | **55** | **0.18** |
| **20** | **75** | **0.27** |
| **30** | **83** | **0.36** |
| 50 | 92 | 0.54 |
| 70 | 95 | 0.74 |
| 100 | 100 | 1 |

The energy efficiency of the server system has improved during recent years. Table 12 shows the energy efficiency of a rack-mounted server system from Fujitsu[88] in 2015 measured with *SPECpower* [SPE 2015]. Eight years after system $II$ measurement (Table 11), the energy efficiency has doubled at low utilization levels $(10 - 30\%)$ because of efficient components, design, and architecture. Even so, there is a constant quest for improvement at a low utilization level.

[87] Energy efficiency: the authors calculate the power efficiency, called energy efficiency, by dividing the utilization level and the corresponding power, both normalized by their peak values
[88] Fujitsu: Fujitsu Limited, http://www.fujitsu.com/global/

Table 12: Server utilization, power, and energy efficiency – system $III$ [SPE 2015]

| Utilization [%] | Power [% of peak] | Energy efficiency |
|---|---|---|
| 0 | 0 | 0 |
| **10** | **32** | **0.32** |
| **20** | **37** | **0.53** |
| **30** | **43** | **0.7** |
| 50 | 54 | 0.93 |
| 70 | 69 | 1 |
| 100 | 100 | 1 |

Reducing idle times is a management strategy. Performance values check and compare systems and their characteristics. Benchmarks are standardized, synthetic sets of applications, which are executed on the system. Performance metrics and values identify how powerful such systems could be. This specified standard is a reference point for comparative purposes. It establishes how efficient a system is designed and operated. A few repeated measurement iterations support statistically significant statements. Data center owners or operators compare the efficiency of key facilities, such as cooling, lighting, or power distribution. An indicator of the HVAC system performance is the ratio between cooling and UPS power [TSX et al. 2003], as shown in Equation (3.14). The UPS output strongly relies on the system utilization.

$$HVAC_{performance}[\%] = \frac{HVAC_{power}\,[kW]}{UPS_{output}\,[kW]} \tag{3.14}$$

The HVAC system effectiveness is calculated to be IT equipment energy divided with full HVAC system energy, which is a sum of the electrical energy for all HVAC components, such as the cooling, fan movement, fuel, steam, and chilled water [WK 2013].

$$HVAC_{effectiveness}[\%] = \frac{IT\ equipment\ energy}{HVAC\ system\ engery} \tag{3.15}$$

HVAC systems are integral elements of data centers and form a part of the total energy consumption of data centers. This is applicable on rack enclosures as well. Table 13 shows the metrics and key performance indicators (KPIs) in various domains, such as data centers. The IT or load density[89] of server equipment or server utilization factors are examples of key performance factors in rack and server domains. On the other hand, time to market conditions such as the transaction and response time are a part of the service level agreements of data centers. Servers are analyzed not only concerning power consumption but also concerning throughput, such as the bandwidth [GB/sec], performance [GFLOPS/TFLOPS], or total numbers of instructions or cycles. Appendix A3b provides an enhanced table of metrics in various

[89] Density: watts per square feet [W/sf]

domains. The Top 500[90] benchmarks the energy-efficient supercomputer in the field of high-performance computing (HPC) using *LINPACK*[91]. *JouleSort*[92] and *SPECpower* similarly benchmark servers concerning their computing efficiency by measuring the system power/energy, performance, or time while processing an application.

**Table 13: Metrics and benchmarks (performance, power/energy, and efficiency) in various domains**

| Domain | Metrics | Benchmarks |
|---|---|---|
| **Data center** | Power usage effectiveness (PUE), airflow efficiency (AE), cooling system sizing (CSS), data center cooling system efficiency (CSE), carbon usage effectiveness (CUE), data center workload power efficiency (DWPE), coefficient of performance (COP) | Calarch, Comis, DoE-2, EnergyPlus, Genopt |
| **Rack enclosure** | IT or server equipment load density (W/sf), SWaP (space, watts, and performance), PDU losses, return temperature index (RTI), rack cooling index (RCI), beta index ($\beta$) | |
| **Server system, component** | IT equipment utilization (ITEU), IT equipment efficiency (ITEE), utilization (load) factor, server utilization, green computing performance index (GCPI), peak performance (GFLOPS, TFLOPS) | Green 500, SPEC CPU, SPECpower, LINPACK, STREAM, JouleSort, Server Efficiency Rating Tool (SERT) |
| **Chip** | Performance counter, instructions or cycles, thermal resistance | Latbench, Micro-Benchmarks |

Each domain includes performance indicators with magnification by a factor of 10, starting from the chip moving towards the data center's domain. Because of their performance and energy usage, the carbon footprint production of each domain is proportional to the magnification factors. Meanwhile, the servers are in operating mode, producing 80 up to 90 percent of the entire carbon footprints[93] [Fuj 2010].

Power is a key factor across all domains, as shown in Figure 28. Costs, either operational or expenditure, are the main factors in data centers and build limits for the design process and investments. A high computing performance reduces working time needed for given services. On the other hand, increasing maximum power consumption leads to growing thermal costs. Thermal considerations have less impact in the chip domain in comparison to server or data

[90] Top 500: http://www.top500.org/project/linpack/
[91] LINPACK: https://software.intel.com/en-us/articles/intel-math-kernel-library-linpack-download
[92] JouleSort: http://sortbenchmark.org/
[93] Total carbon footprint: raw material, manufacturing, transport/distribution, assembly, use phase, recycling, and disposal

centers. Airflow and humidity are listed separately, as shown in Figure 28, because they are not considered in most performance metrics for chips and servers. All mentioned aspects, thermal, power/energy, utilization, carbon footprint, airflow/humidity, and costs have same goals towards performance and efficiency but differ in their approaches.

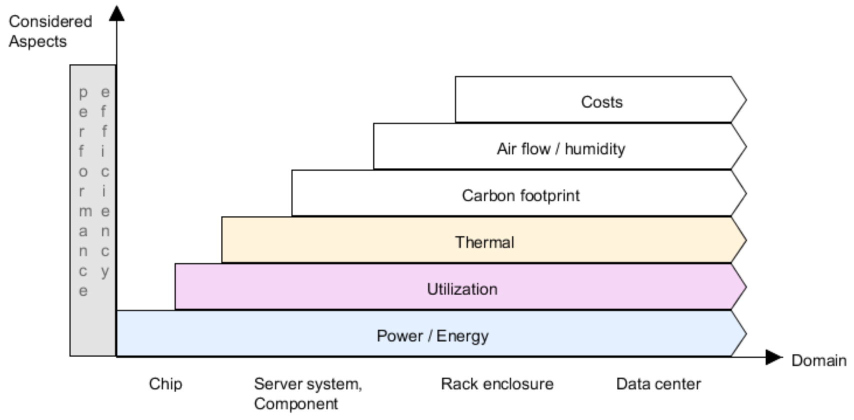


Figure 28: Considered aspects in various server domains

The performance and efficiency aspects correspond separately to thermal, utilization, power/energy, and time but are combinable. The performance metrics indirectly express the systems' behavior in various domains, while the busy and idle thread metrics present the utilization aspect of a resource, for instance. The processor provides computational power to offer the highest throughput, but on the other side, it generates extra heat. The high numbers of exceptions or errors are the reason that a task execution may take longer, and the costs increase. The performance is considered in each domain, but is shown as a single aspect in the figure to depict the huge amount of the metrics.

Additionally, each aspect also has many metrics, not only performance, to determine the resources' behavior. The power losses (in the PSU, PDU, or UPS) are common metrics at the server, rack, and data center level. The utilization levels are fundamental metrics in the component level. At low level, the thermal resistances define the temperature behavior. On the one hand, the metrics are a domain; however, they are included at each considered aspect. If we integrate the metrics in Figure 28, the objects overlap, and they are illegible. Therefore, the aspects completely move to the y-axis. The x-y-position of the various domain-specific metrics and indicators in Figure 29 shows the direct relation between the server domain and the considered aspect. This thesis focuses on thermal, power/energy, utilization, and performance indicators, which are key factors for server systems and components. The other indicators such as the carbon emission, airflow / humidity, equipment density, or costs are not considered.

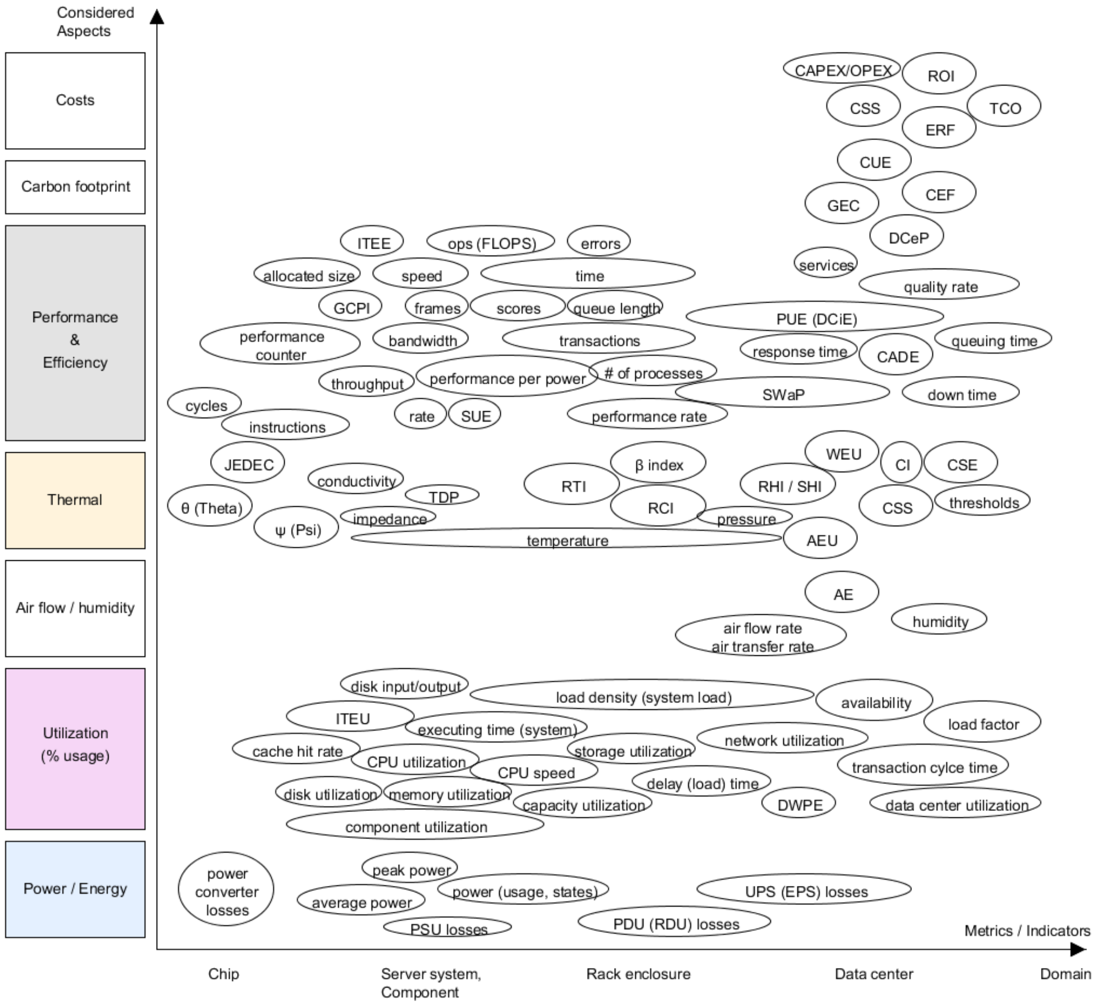


Figure 29: Metrics and considered aspects in various server domains

Server domains with performance, efficiency, or utilization aspects contain most metrics and indicators. Data center metrics focus on costs and performance. In this thesis, the server domain is the main focus and is defined as a system domain. The environment abstracts external system influences, such as rack enclosures and data centers. The physical domain clusters the internal chips, transistors, gates, logic, and circuits. Table 14 provides a short overview and examples of the system domain definition.

Table 14: Focused domains for thermal, power, and performance indicators

| Domain | Server system domain | Focus |
|---|---|---|
| **Data center** | Environment | Hypervisor / virtual machine monitor, infrastructure equipment, operating system level, software level |
| **Rack enclosure** | Environment | Network equipment, storage equipment, server |
| **Server system, component** | System | Components (processor, memory, bus), electronic system level (ESL), connectors |
| **Chip** | Physical (Chip) | Circuit, transistor, gate, logic |

Analyzing the server system domain, server definition (Section 2.4.1), and the modeling domains (Section 3.1.4) lead to the following server type description, classified in Figure 30:

I. The physical server system description, including:
   - A1. Electrical descriptions for the power/energy, thermal, performance, and efficiency calculation also within chip level
   - A2. Thermal equations for heat transfer
   - A3. Mechanical definitions that define the physical system characteristics: e.g., the largest amount of components or the total number of components currently used, such as memory modules, etc.

II. Interfaces (Peripheral Component Interconnect), operations, and communication processes to handle jobs and their related descriptions are part of the logical level.

III. Hardware and software designs define the topology, hierarchy, generation, or architecture views of server systems. Furthermore, they include components and connectors to support logical description.

IV. The conceptual description of the server system defines data types, general/abstract functions, workflow (control flow), communication, databases, processes, or controllers.

V. The structural or architectural description is part of the conceptual and contextual because it includes the topology, entities, coupling, attributes, or component relationships.

VI. The customization is part of the contextual and the external domain. Herein, the customer defines the server system's hardware and software configuration, such as the operating system (OS) type, or BIOS/UEFI settings.

VII. The usage context defines the server system environment. The server system inputs are the ambient temperature, utilization level, and server configuration.

VIII. The model's purpose (Section 3.2) is the considered aspects (Section 3.3), which are partly described within the contextual and the external level.

IX. Measurement data, such as benchmark results, device tables, or data sheets are external conditions for system development and design.

X. Constraints are external sources because vendors define thermal limits for the components. The system designer and company predefine some rules, limits, laws, policies, schema, or terminologies.

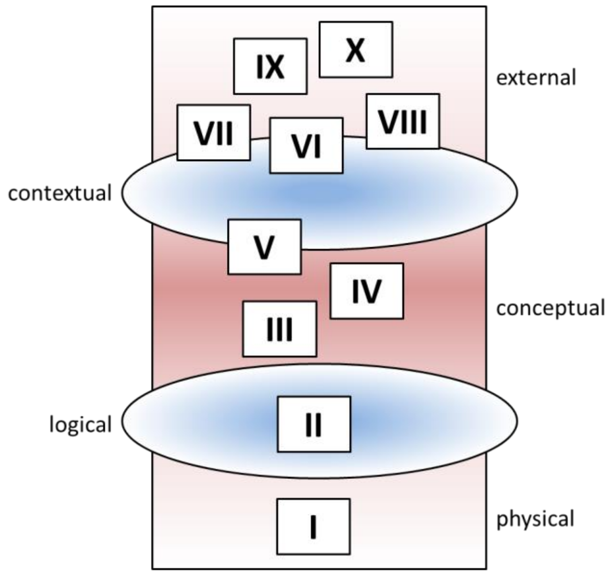


Figure 30: Definition of server modeling domains using server system definition, enhanced model from [Osi 2010]

If we add the considered aspects, the planar abstraction will become a three-dimensional graph, wherein the x-z-plane represents Figure 30 partly at the server system domains. Each aspect is taken into account and has a model in a server system domain, represented in the x-y-plane. Furthermore, the y-z-plane represents these aspects within the server-modeling domain.

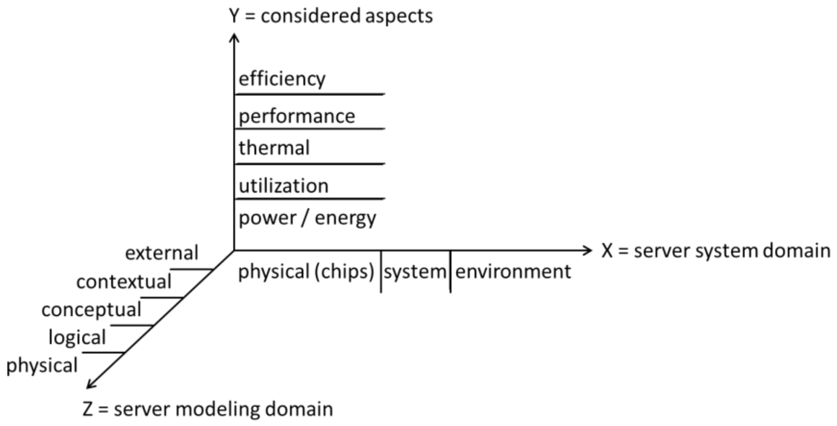


Figure 31: 3-dimensional space for diverse models

The next sections describe the fundamental algorithms and approaches initiated from the various model aspects, which can be classified in the server system and server-modeling domain. The models are substantial for the management techniques and the optimization algorithms.

## 3.4 Power and Energy Algorithms and Approaches

The following power and energy models are described along the x-axis at various server system domains. The white-, gray-, or black-box models are domain-specific because of the inner data or behavior. This section presents various models beginning with the full information, such as instructions up to the least well-known internal data. Several prototypes of low-level power devices apply cycle-accurate approaches.

### 3.4.1 Physical (Chips) – Server System Domain

The power and energy model approaches and algorithms are usually defined as a physical or logical description of the chip or component level. Chip level approaches define their electrical behavior at the transistor, circuit, gate, register, or transfer level. The interior component structures, operations, and processes are known because of existing data on the geometry, design, and topology of the circuits, characterized by real hardware measurements. Monitoring, profiling, or tracking the software and corresponding power values give the ability to create power metrics. If the data about the natural linking of the devices, the used blocks providing service, their operations, the data transfer at register-transfer level (RTL), and the chip's functions and signals are obtainable, all internal system data is visible, and thus it is a white-box model.

***White-box Approaches (Instructions)***

We categorize the white-box power models into logical and electrical[94] descriptions. The logical description includes instructions or activities to define the system behavior. The logical-based approach uses physical measurement techniques to obtain power values while executing associated and disassembled instructions running on the component. In this case, chip level details are not required. In contrast, simulation-based approaches need information about the chip's micro-architecture, which relies on functional or algorithmic levels.

In [TMW 1994], fundamental research is done for measurement-based power analysis techniques. The average power $P$ is defined by the average current $I$, and the voltage supply $V_{CC}$. The energy $E$ is the consumed power over the execution time $T$ of an instruction, operation, given task, or software. Moreover, the execution time of a software consists of the clock cycle's $N$ and the corresponding clock period $\tau$, as shown in Equations (3.16) and (3.17).

[94] Electrical description: gate and circuit

$$P = I * V_{CC} \tag{3.16}$$

$$E = P * T, T = N * \tau \tag{3.17}$$

Individual instructions have their specific energy consumption. The total energy is determined by the base cost of one cycle multiplied by the full number of cycles for the instruction. This approach is called instruction-level power analysis (ILPA). Tiwari considers the inter-instruction effects in the power model. Furthermore, Tiwari et al. analyzed the software power consumption on the basis of the instruction level on a digital signal processor [TMW et al. 1996]. They found that the state changes (switching activities) in the circuit generate more overhead in comparison to the single instructions at each time. These inter-instruction effects are considered within assigned costs. Therefore, the author refined the total instruction-level energy model whereby the overall energy $E$ is the total of:

1. "base costs, $B_i$, of each instruction, $i$, weighted by the number of times, $N_i$, it will be executed, [TMW et al. 1996]"
2. "the circuit state overhead, $O_{i,j}$, for each pair of consecutive instructions, $(i, j)$, weighted by the number of times, $N_i$, the pair is executed, TMW et al. 1996] "
3. "the energy contribution, $E_k$,of the other inter-instruction effects, $k$, (pipeline stalls and cache misses) that would occur during the execution, [TMW et al. 1996] "

Equation (3.18) shows the total energy calculation. The power analysis uses the entire instruction set architecture (ISA) to characterize single instructions.

$$E = \sum_i (B_i * N_i) + \sum_{i,j} \left(O_{i,j} * N_{i,j}\right) + \sum_k (E_k) \tag{3.18}$$

Most of the measurement-based, instruction-level power models refer to Tiwari's method. The authors of [NKN et al. 2002] also proposes a measurement-based, instruction-level power model, but on a current sensing circuit. Each clock cycle and related instantaneous currents are monitored and measured during the execution of instructions such as ADD, AND, or MOV. In the work of [RHH et al. 2005] the analytical power model is applied on a processor for a low-power embedded system on a chip (SoC). In addition, they characterized the instructions in five main groups.

Another adaptation to the Tiwari's method is done by [SIC 2003]. The power consumption of the components includes the dynamics, and the static power defined in Equations (3.19), (3.20), and (3.21). The dynamic or switching power is calculated by the whole average capacitance[95] $C_L$, the supply voltage $V_{DD}$, and the operating frequency $f$. The capacitance depends upon the software execution per clock cycle. The static or leakage power is the product of voltage $V_{DD}$ and the leakage current $I_{leak}$ through the circuits. The fixed power consumption results from bias currents, junction currents, or gate tunneling, for instance.

[95] Total average capacitance: sometimes also defined as effective (switched) capacitance $C_{eff}$

$$P = P_{dynamic} + P_{static} \tag{3.19}$$

$$P_{dynamic} = C_L * V_{DD}^2 * f \tag{3.20}$$

$$P_{static} = V_{DD} * I_{leak} \tag{3.21}$$

In addition, the authors of [HXL et al. 2002] added a factor $\propto$ to the dynamic power definition. In the case of complementary metal-oxide semiconductor (CMOS) or very large-scale integration (VLSI) circuits, the factor $\propto$ describes the switching activity ratio, and $C$ is the physical load capacitance [JGM 2003], as stated in Equation (3.22).

$$P_{dynamic} = C * V_{DD}^2 * \propto * f \tag{3.22}$$

Landman [Lan 1996] classifies the architectural power models in complexity-based and activity-based models and describes the static as well as dynamic activities at the behavior-level. On the one side, the numbers of functional blocks, which are equal to used gates for a specific function, define the chip architecture's complexity. On the other side, the authors of [Naj 1995] found that the consumed power relies on the input and output entropies of the functional blocks. Three high-level approaches in the industrial use case, considered in [FCM 2014], mainly analyze FPGA power consumption. They offer an overview about basic hardware design flow beginning at the physical level, to gate level, register-transfer level, and up to system level.

Other approaches include the micro-architectural structure of the circuits. The authors in [KAM et al. 2002] give an overview about micro-architectural power methods. Hence, the authors of [LS 1994] propose a power model that includes the logic gate functions, the capacitance, the latches coming from flip-flops, the cell structures of a memory module using cache lines, the row decoder, the column selector, or intermediate interconnections such as buses or wires. In fact, the approach requires library information[96] about the circuits at the gate level. In [WJ 1996], they present an analytical model for an on-chip, direct-mapped cache. The authors focus on access and cycle times at a cache array. Moreover, they consider the parameter (cache size, block size, or width), the cache organization parameters (bit lines, word lines, or array size), decoder structures at gate level, comparators, multiplexers, drains, resistances, or capacitance. An overview about the various types of capacitance and their equations are in [BTM 2000, NKB et al. 2004]. In [HXL et al. 2002], the authors consider the gate and transistor capacitance within an architectural-level model, such as, using the first-in first-out (FIFO) method. Another execution-driven, cycle-accurate power model of a memory system, including various access stages, is addressed in [YVK et al. 2000]. The output is traced cycle-by-cycle at the register-transfer level. They proposed a transition-sensitive energy macro-model. A macro-model considers a function or unit as a black-box model and is only aware of

[96] Library information: composition rules and technical description of basic building blocks of electronic functions (layout, schematic, logics, shapes, or symbols)

the power consumption of a subroutine within an internal module[97]. The authors of [LJ 2003] present a linear model, including the power interest metrics $c_j$ and macro-modeling coefficients $w_j$.

$$P = \sum_j w_j * c_j \tag{3.23}$$

A control flow graph defines the correlated paths of the functions or subroutines. The authors of [LRR et al. 2004] address another early-stage power model at the RTL or lower level, based upon a cycle-accurate functional description. It uses system activities, transition states, data paths, or buses.

The previous physical and logical models (white-box models) are instruction-level power modeling approaches, which need detailed design data (architecture, structure, transitions, execution units, registers, or activities) about the circuit level in the physical system domain. These approaches have restrictions because of their low abstraction level. Usually, we need an analysis[98] of the chip to build a model unless earlier data are available. Therefore, the software or instruction[99] sequences run on a real chip and in parallel, the customer receives the data. The circuit- or gate-level models become infeasible and extremely slow because of the simulating complexity of a large software application [LJ 2003]. In [YVK et al. 2000], the authors propose the memory system's power model in the physical domain. These techniques are too inefficient for system-level design. However, some unknowns such as the bus architecture, read/write access, context switches, or the total number of bus transactions must be predicted. Furthermore, the actually used instructions on the chip are unknown before the customer or developer assembles the chip. In the design phase, the developer determines the suitable chip for a specific demand to give proper support. For instance, a network chip[100] should support the data rate of 10/100/1000 megabit per second, has a package size of 81mm², and maximally use a two-volt supply voltage. The designer selects the chip by charging the minimal costs. In sum, the energy and costs are major design decisions in comparison with the chip architecture, structure, or activity. In addition, the design and layout of the system may have not been specified. Various design parameters, such as circuit styles, clock strategies, word lengths, signals, or layout techniques are impossible to characterize in the early design phase. Additionally, we cannot guarantee the detailed circuit-level information availability.

[97] Internal modules: adders, registers, multipliers, or controllers
[98] Analysis: measuring, profiling, or tracking of software, instructions, or activities
[99] Instructions: assembly-level
[100] Network chip: Ethernet PHY chip, physical layer transceiver

### 3.4.2 Components and System – Server System Domain

***Gray-box Approaches (States and Transitions) – Processors***

Besides the instruction-level approaches, the hardware monitors[101] trace micro-architectural events, such as accesses, and switching activities, such as cache miss times. A power model uses the counter-based heuristics that reflect the hardware activities. The authors of [JM 2001] distinguish between hardware performance counters and power-relevant events of microprocessors. In their approach, professional monitoring tools trace the processor's performance counters and events. The processor's functional units measure and verify the power of an executed application. Bellosa analyzed the floating-point or integer instructions on a processor and the event correlation, for instance [Bel 2000]. [SBM 2009], [RAK et al. 2013], and [LSQ et al. 2014] provide power models using performance counters or hardware events, such as instructions per cycle, fetch counters, miss/hit counters, stalls, retired-instruction counters, clock logic, data path, cache, or other events to choose the correct operating condition. Another approach considers per component $c_i$ the access rate $ar$, the architectural scaling $as$, the maximum consumed power $mp$, and the conditional clock power $ngcp$ for the component power, outlined in [IM 2003]. The authors analyzed a Pentium 4 processor and its functional units. They used 24 performance event metrics for the model of the processor power. The total power is the sum of the idle power and the power of all components, as shown in (3.24). The system power varies between 30 and 50 watts while executing several benchmarks. Their model predicts the power with a variation about three Watts. The authors of [BGM et al. 2010], applied this approach to the power of the processor cores.

$$P_{total} = P_{idle} + \sum_i (ar\,(c_i) * \; as(c_i) * mp(c_i) * ngcp(c_i)) \tag{3.24}$$

Those power model approaches are sufficient if real hardware is available. Through these performance counters, they are able to generate power values for a specific application. The hardware performance counters provide detailed processor data in comparison to events coming from the operating system or application. However, if the data on an application or their operating type[102] is not available, we cannot estimate or predict the power. The application may be non-deterministic and vary over time throughout diverse executions. Bus transactions[103] from other components may influence the access rate. On the other hand, the processor is still in development or only a prototype is shipped to the server system vendor during the design phase. Therefore, the performance counters are either not traceable or do not present the last version. In addition, the processor type limits the heuristics, and with them, the performance counters. Furthermore, the measurable events simultaneously deviate, as analyzed in [JM 2001]. One reason is the various CMOS types used by the application. The

[101] Hardware monitors: performance counters which are a group of special registers
[102] Operating type: integer, floating-point, arithmetic, or logic
[103] Bus transactions: data activities on the front side bus (FSB) are reads, writes, or pre-fetches

performance counter approaches are not valid for novel multi-core processors since new processor architectures, such as extra cache levels or clustered functional units are introduced. In [Ben 2010], the authors propose that various caches have a power proportion of approximately 30 percent of the entire dynamic power profile of a processor. Two other major parts are the clock and the functional units.

Moreover, the event-driven model attempts to generalize the instructions and performance counters by using resources and events. Each resource within the system is a state machine wherein the state is component behavior abstractions, including data about the power behavior of a block, block interactions, or environmental data. An event leads to a state change, also called a transition. Section 3.1.2 shows the formal definition. A transition is marked with costs, times, or events within the state machine. The power of a resource is associated with the actual state. Hence, a state is marked with a power value or function. Benini [BHS 1998] presents various power state machines (PSM) including the operational states for a display, a disk, a memory module, and an electronic clipboard. Three years earlier, Benini [BM 1995] described a state transition graph (STG) as a Markov chain for a simple finite state machine at a high-level abstraction. The Figure 32 illustrates an equal power state machine of a server containing only the two states *on* and *off* on the basis of a formal definition of Figure 20. In this model, the power consumption is zero when the server is *off* and otherwise a function of maximal power and the internal system state if the server is working. The power events[104] trigger the diverse transitions between the two server states.

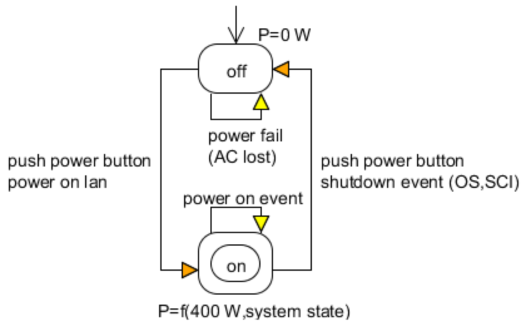


Figure 32: Simple power state machine of server (on, off)

[104] Power events: customer shutdown via operating system or automatic shutdown event via system control interrupt (SCI)

The advanced configuration and power interface (ACPI) specification[105] defines the system states into:

- Global system states $(G)$
- Device power states $(D)$
- Sleeping states $(S)$
- Processor power states $(C)$
- Device and processor performance states $(P)$

Appendix A3a gives an overview of all global system states specified by ACPI [HIM et al. 2013]. The global system state defines, for instance, if the system is in mechanical off $(G3)$, soft off $(G2)$, or the working $(G0)$ state. The device power states are adequately distinguished in off or full-on. The sleeping states $(S0 - S5)$ include the power and latency. The processor power $(C)$ and performance $(P)$ states are interpreted by processor vendors such *Intel* [CJ 2010], which defines and extends the states as follows:

- System sleep state $(S)$
- Microprocessor and package idle state $(C)$
- Microprocessor performance and operational states $(P)$
- Microprocessor throttle states $(T)$

An overview of the internal processor states shows Table 15, Table 16, and Figure 33. The states consists of $N$ number of sub-states, whereby $N$ is any natural number $N_0 = \{0, 1, 2, 3, \dots\}$. The related power management approaches are described in Section 3.4.2. For instance, the throttling can reduce the noise of the fans because a passive cooling is enough after the power reduction.

**Table 15: *Intel* processor states $(S)$ – an overview [CJ 2010]**

| *Intel* States | State | Description |
|---|---|---|
| **System sleep states** | $S$ | **Sleeping state** |
| | $S0$ | Full on |
| | $S1 \dots S5$ | Sleep states |
| | $S3$ | Suspend-to-RAM |
| | $S4$ | Suspend-to-disk |
| | $S5$ | Soft off |

[105] ACPI specification: also defines the transitions between the states

Table 16: *Intel* processor states $(C, P, T)$ – an overview [CJ 2010]

| ***Intel*** **States** | **State** | **Description** |
|---|---|---|
| **Microprocessor and package idle state** | $\boldsymbol{C}$ | **Processor power state** |
| | $C0$ | Executes instructions active, operational |
| | $C1 \dots C7$ | Not fully active |
| | $C1, C1E$ | Halt |
| | $C3$ | Sleep |
| | $C6, C7$ | Deep sleep |
| **Microprocessor performance and operational states** | $\boldsymbol{P}$ | **Performance state** |
| | $P0$ | Maximum performance turbo |
| | $P1$ | |
| | $PN$ | Energy efficient state |
| **Microprocessor throttle states** | $\boldsymbol{T}$ | **Throttle states** |

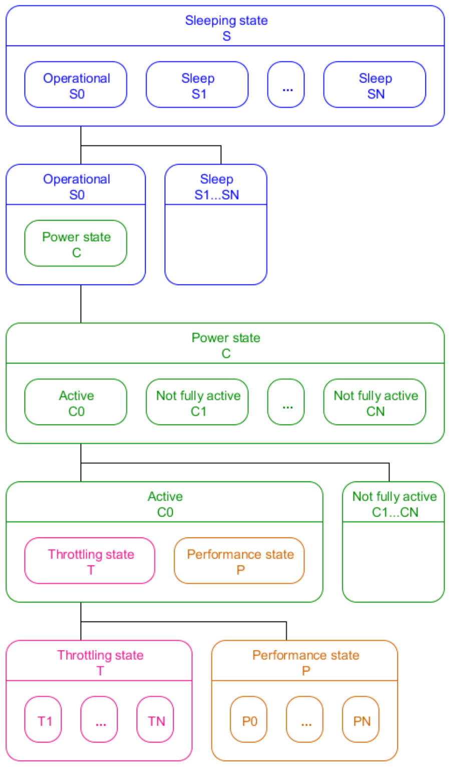


Figure 33: *Intel* processor states – a graphical representation [CJ 2010]

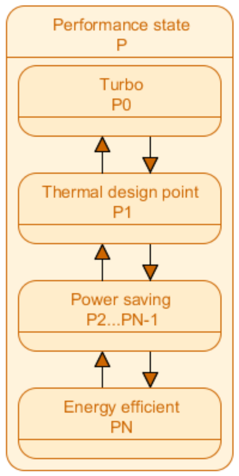


Figure 34: *Intel* processor $(P)$ states [CJ 2010]

The authors of [BJ 2003] use a power state machine for each core at a PowerPC. The developed framework calculates the minimal and maximal power, which relies on a reachable set of power states. In addition, they analyze the impact of various parameters and configurations. Their algorithm includes the spreadsheet models, which will be outlined.

The power model of the state-based approach[106] consists of three major power consumptions, whereby $t_{ij} \in T, s_i, s_j \in Q,\ i, j \in N_0, N_0 = \{0, 1, 2, 3, \dots\}$.

1. The leakage power consumption $P_l(s_i)$ exists for all appearing states $s_i$[107].
2. The power consumption $P_s(s_i)$ is defined as the average power for all operations in each state $s_i$.
3. The transition power[108] $P_t(t_{ij})$ is the dissipated power for a transition $t_{ij}$, which switches the state $s_i$ to another state $s_j$.

In conclusion, the power model sums up the power values for each used state and all occurring transition power, as stated in the following equations.

$$P_{leakage} = \sum_{i=1}^{m} P_l(s_i) \tag{3.25}$$

$$P_{states} = \sum_{i=1}^{m} P_s(s_i) \tag{3.26}$$

$$P_{transitions} = \sum_{i=0}^{n} \sum_{j=0}^{m} P_t(t_{ij})\, , whereby\ i = j\ is\ negligible \tag{3.27}$$

$$P = P_{states}(used) + P_{transitions}(used) + P_{leakage}(used) \tag{3.28}$$

The state-based approach is applied wherever the resource's state changes are detectable. The model defines the power behavior in an abstract manner and does not consider the system's inactivity or corresponding overhead in comparison to Tiwari's method. The approach only considers the power behavior of the resource and the transitions. The lookup tables (LUT) integrate the power models, for instance, as a hardware-specific macro. The authors of [CBB et al. 2010] focused on the processor's power domains to support a set of various power modes where the power domains come from the diverse voltage regulators within the system on chip.

The clock frequency of the respective resource formulates the power states. Current processors or memories today operate at various frequencies in the interval $[f_{min}, f_{max}]$ at runtime. A related power value within the interval of $[P_{min}, P_{max}]$ exists for each frequency. Equation (3.29) shows the power-frequency relationship, proposed by [UKI et al. 2010]. The authors found a quadratic power model which consists of the minimal power $P_{min}$, the weight factor $\propto$, the operating frequency $f$, and least frequency $f_{min}$ of the processor.

[106] State-based approach: formal definition can be found in Section 3.1.2
[107] State: can consist of several sub (internal) states
[108] Power: it is precisely the energy because of consumed time for a state change

[GHD et al. 2009] present another similarly quadratic relationship. The minimal power, which is more than 30%, and the power trends are shown in Table 10, Table 11, and Table 12. In [Han 2007], the power model of the processors uses the p-state and the corresponding frequency.

$$P(f) = P_{min} + \propto * (f - f_{min})^2 \tag{3.29}$$

In [TDM 2011], the researchers analyze a multi-core processor[109] because of the number of cores and neglected active frequency in previous approaches. The authors propose a power model with a linear association of the processor‘s power consumption and the number of working cores. They add the frequency relationship and power regression factors to predict the power consumption of various application types. In [MAC et al. 2011], they present a linear and non-linear regression model, which considers multi-core processors, including hidden device states. Equation (3.30) shows the additive regression model, which is solved by calculating the least square solution for the parameters $\beta_0, \beta_1, \dots, \beta_n$. The vector $y = [y_1, y_2, \dots, y_k]$ contains the power measurements; the vector $x_i = [x_{i,1}, x_{i,2}, \dots, x_{i,n}]$ includes the normalized vector of measurements on the $n$ variables, and $\varepsilon_i$ is the numerical noise.

$$y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \dots + \beta_n x_{i,n} + \varepsilon_i \tag{3.30}$$

The evaluation results show that a linear and non-linear model of the processor differs from 10 up to 150 percent, such as, using the interchangeable configuration for the same dies. [BGM et al. 2010] and [BM 2012] present core-based power models, such as in Equation (3.31). The power consumption $P_c$ of a core $j$ is total for all $n$ cores. The entire power consumption is the sum of the power dissipated by each single core.

$$P_n = \sum_{j=1}^{n} P_c(j) \tag{3.31}$$

The processor model of [KJC et al. 2014] considers the number of active cores, the operating frequency, the number of cache misses, and accesses. The processor power is the sum of the dynamic power, the static power, and the cache power, whereby $c$ denotes the number of working cores. The dynamic power consists of a constant $k$ and the constant factor $\beta$, which depends upon capacitance, the supply voltage $V$, and the activity factor of the processor. The static power is the product of leakage current and supply voltage. The cache power multiplies the number of cache accesses $N$ and a constant $\varepsilon$.

$$P_{processor} = P_{dynamic} * c + P_{static} + P_{cache} = \beta(kV)^3 c + \gamma(kV) + \varepsilon N \tag{3.32}$$

The authors in [BM 2012] suggest a multi-core processor model, including resource sharing. They further analyzed the chip-level, and the die-level constraints as well as the communication between the cores. Secondly, they extended their core-level model by adding active core power consumption and inter-core communication. The resulting power model is

[109] Multi-core processor: *Intel Xeon CPU E5540*

based on the core-based utilization level. The entire processor power, in Equation (3.33), consists of the chip-level mandatory component power $P_{mc}$, the communication power between dies $P_{inter\_die}$, and the die-level power $P_{die}$. The die-level power includes the core-based power $P_n$, the inner die-level power, and the off-chip caches power. In [BM 2012], the authors describe information that is more detailed.

$$P_{processor} = P_{mc} + P_{inter\_die} + P_{die} \tag{3.33}$$

The peak power consumption of the processor core is analyzed in [BCS et al. 2012], produced by the functional pattern's execution. The authors divide the simulation-based approaches into switching activity (SA), weighted switching activity (WSA), transient analysis (TA), and post layout analysis (PLA), which refers to the abstraction level and the domain. Furthermore, they provide a comparison of the accuracy and the runtime of these approaches.

Moreover, the work of [FWB 2007] shows a model that uses the processor utilization reported by the operating system. The dynamic power of components besides the processor and memory is negligible because the power values are less than 30%. Benini addresses in [Ben 2010] the power consumption trend and power density trend, where the leakage and the dynamic power increase linearly with the power density. Additionally, they state that the supply voltage $V_{DD}$ decreased constantly by approximately 0.8-fold with each step of shrinking {250, 180, 130, 90, 65, 45} nm technology, but the energy per device decreased by 50 percent. Benini also found that the active power increases by 50 percent when the frequency doubles.

All simulation-based models have in common that a resource such as a processor must be available. The power models work with measurement results using performance counters, present frequencies, core activities, or access rates. Hence, the approaches use data about inner designs, structures, states, connectivity, or parameters. The models characterize the components and improve the power estimation because of higher detail and accuracy. In comparison to the chip domain, data about internal structures are necessary to gain, but not to every single, last, or full detail. For that reason, the approach is a gray-box view of the system. The white-box and gray-box approaches differ in their accuracy and speed. The instruction-level approaches are more accurate in comparison to the gray-box approaches but at the same time very slow. Additionally, the more accurate models are very time-complex[110], and if they only present a small part of the overall power consumption, this means a waste of effort. The main aim must be to optimize the operating costs.

[110] Time-complex: high design and computational effort

***Gray-box Approaches (States and Transitions) – Memory and Disk***

In addition to the modeling of processors, other devices and components are considered, such as memory modules and storages. In [LR 1996], they introduce models for computational elements. Equation (3.34) shows the *PowerPlay's* dynamic power model of a static random-access memory (SRAM) module. "The switching capacitance,"…", is a function of the word-width (bits) and the number of words, [LR 1996]." They divide the swing capacitance into $C_{partialswing}$ and $C_{fullswing}$, which takes the related swing voltage[111] $V_{swing}$ at a frequency $f$ into account. The authors of [NKB et al. 2004] consider a bit-line capacitance factor per module by using an overall swinging delta voltage. In contrast, [ZYY et al. 2011] proposes a power model considering the voltage and the current only.

$$P_{dynamic} = \propto * \left[C_{fullswing}V_{DD}^2 + C_{partialswing}V_{swing}V_{DD}\right] * f \tag{3.34}$$

[HCE et al. 2011] describe a memory power model that quantifies the dynamic random-access memory (DRAM) with respect to operational and background power. The active memory operations, especially all accesses or data transfers, are the essential part of the dynamic power; on the contrary to the power models of the processors, the operating voltages, frequencies, and associated states are elements of the background power. The background energy $E_{background}$, as shown in Equations (3.35) up to (3.40)[112], is the sum of the consumed energy for self-refreshing[113] the memory module, the energy to precharge the data, and the overall energy produced by the data read/write operations[114]. In [LEU et al. 2010], they adopt the model using weights for the operations, such as activate, read, or write access. The authors in [LZZ et al. 2007] state a model that uses the throughput of the read and write access for the static power estimation.

$$E_{background} = E_{self\_refresh} + E_{precharge} + E_{read\_write} \tag{3.35}$$

$$E_{self\_refresh} = (P_{self\_refresh} * t_{self\_refresh}) \tag{3.36}$$

$$E_{precharge} = E_{precharge\_fast\_powerdown} + E_{precharge\_standby} \tag{3.37}$$

$$E_{precharge\_fast\_powerdown} = (P_{CKEL} * t_{CKEL}) \tag{3.38}$$

$$E_{precharge\_standby} = (P_{CKEH} * t_{CKEH}) \tag{3.39}$$

[111] Swing voltage (CMOS): output voltage range (rail-to-rail) usually from $V_{DD}(+)$ to $V_{SS}(-)$
[112] CKEL/CKEH: clock enable low/high
[113] Self-refresh: activating the module via row access
[114] Data read/write: column access

$$E_{read\_write} = E_{read} + E_{write} \tag{3.40}$$

$$E_{read} = (\frac{E_{read}}{ops} * Bandwidth_{read}) \tag{3.41}$$

$$E_{write} = (\frac{E_{write}}{ops} * Bandwidth_{write}) \tag{3.42}$$

The authors of [HJZ et al. 2008] focus on background power in the same manner, but using a state transition model. Their model considers the memory module design, including interfaces, buses, pins[115], and logics. They found that the consumed power is proportional to the bits that are read and written. Furthermore, the power depends on the data width because of the involved number of devices for each access. The authors of [JCX 2008] address a typical state machine of DRAM, which is equally valid for various random-access memories[116]. The authors present the models for background, activate, read/write, and total power, including the currents and voltages of the memory module. A similar power model is done in [Qia 2011], whereby the idle, the active, and the standby state specify the current power state and the spin down, the spin up, and the seek is the related transition.

In [XTB 2007], the authors introduce an analytical and empirical model for SRAM structures. The energy model of the analytical SRAM array bit line, shown in Equation (3.43), has the supply voltage $V_{DD}$, the total capacitance $C_{bitline}$, and the voltage swing $V_{swing}$ of the bit line, whereby the capacitance $C_{bitline}$, for example, consists of pre-charge circuit capacitance or column-select circuit capacitance.

$$E = C_{bitline} * V_{DD} * V_{swing} \tag{3.43}$$

In [KIM et al. 2011], the authors define a power model based upon the method[117] described in [Mic 2007] for double data rate, synchronous dynamic random-access memories ($DDR3$). In this approach, the termination power is additionally considered. Furthermore, the model uses the percentages of cycles in the various states. The complete energy model is defined by the consumed total power at a specific frequency for a given execution time, shown in Equation (3.44).

$$E_f = P_{total}^{f} * T_{execution} \tag{3.44}$$

[115] Memory pins: data, control, or address
[116] Memory kinds: SRAM, SDRAM, or DRAM (DDR)
[117] Method: memory module calculation of Micron

In the work of [DBN et al. 2005], a memory module simulator called *DRAMsim*[118] uses the previous method. It calculates the power of various memory types, such as $DDR$, $DDR2$, $SDRAM$, and fully buffered DIMMs (FB-DIMM). The simulator uses a detailed cycle-accurate timing model to support four different transaction policies[119]. It is possible to build a power model by using the number of read/write cycles and the relative time at various states.

Power models for hard disk drives (HDD) and solid-state drives (SSD) work in the same manner. The main characteristics are the average execution time for reading or writing operations, the file size, and the number of concurrent processes, as stated in [IIE et al. 2011, IAE et al. 2011]. The total power consumption of hard disk drives is the sum of power dissipated in each operating state for a certain time. Usually, the power is divided into a spin, start, idle, and access, as shown in Equation (3.45). The total power consumption depends upon the number of timeouts, activated by the number of disk accesses within the entire system, as given in [Gre 1994]. The authors of [SLU 2010] additionally consider the bandwidth of the disks, inner disk parameters[120], or the number of disks within an array[121].

$$P_{disk} = P_{spin} * T_{spin} + P_{start} * T_{start} + P_{idle} * T_{idle} + P_{access} * T_{access} \tag{3.45}$$

At the approach of [MPL 2009], the disk model uses two different request types to estimate the power consumption. In this method, they trace a real-time streaming workload, such as data transfers[122] or seek operations[123]. In [GSI et al. 2002], they propose a state-based power model.

The storage power models need access patterns, such as read and write operations and related time data. Additionally, the models use state machines with transitions to estimate the power and energy consumption. The operations, types, and corresponding device states are known. These techniques use behavioral and cycle-accurate functional level models.

The gray-box approaches are functional models that specify a component or behavior in a correct manner. They describe the system’s reaction to an external event, and the related state machines are abstract event simulators of the system [Ben 2010].

[118] DRAMsim: http://www.eng.umd.edu/~blj/dramsim/
[119] Transaction policies: first come first serve (FCFS), read or instruction fetch first (RIFF), bank round robin (BRR), and command pair rank hopping (CPRH)
[120] Internal disk parameter: head switch, track switch, full stroke
[121] Array: merge of physical disks to a logical unit, e.g., redundant array of independent disks (RAID)
[122] Data transfer: read/write operations
[123] Seek operations: physical (address) mapping of logical blocks

***Black-box Approaches (Spreadsheets) – Devices***

We can use a black-box approach even if no other inner data and details are available. The model presents the name and interfaces of a device (*outside point of view*), but no information about the interior behavior or functionality, such as instructions or operations. Therefore, it is applicable for specification and verification purposes in early design phases.

The base for a black-box model is data sheets, which are analyzed to find possible criteria for the power models. We characterize the devices and then the customer selects the characteristics, such as the bandwidth, organization of the device (architecture), cores, or supply voltages. The entire power consumption uses the estimated energy per operation defined in the model under specific constraints, such as the number of accesses of each resource. A web-based spreadsheet is explained in [LR 1996], which considers devices activities. The spreadsheets are tables in which cells contain data, and the function defines the relationships between the column and row variables. The spreadsheets are excerpts of the data sheets and provide power consumption values of each device on the architectural level. The common method is a lookup table[124], which stores the power consumption of a single component whereby the Hamming distance between subsequent input vector pairs indexes each cell [NKB et al. 2004]. Texas Instruments provides power estimation spreadsheets[125] for open multimedia application platform (OMAP) devices.

The black-box approach is not able to estimate power consumption for an explicit instruction. They use maximums or averaged values, "rely on aggregate instruction counts and do not incorporate either time or input data, [DAH et al. 2007]." The switching clock capacitances are not considered for each operation or instruction, but in general are included as a static value within the model. Furthermore, the model is not flexible and has no learning curve because alternative designs, modified technologies, smaller features, or architecture sizes need a fresh device analysis, so a new spreadsheet is necessary [NKB et al. 2004]. The spreadsheet provides the maximal power consumption and does not consider the dynamic power dependent on the frequency or voltage settings. Usually, it is a summation of all components, which states the power consumption at too high a level [BHS 1998]. On the other side, this approach offers a fast calculation in comparison to instruction-based approaches within the early design stages on the architectural level. Even so, the designer has to give some fundamental device facts to model the complex system.

[124] Lookup table: substitutable by simple equation or macro model
[125] Spreadsheets: http://processors.wiki.ti.com/index.php/OMAP3530_Power_Estimation_Spreadsheet

***A Mix of White-, Gray-, and Black-box Approaches – Complete Machine Simulation***

In the academic literature, the simulation approaches regarding the system's power and energy combine and integrate the relevant parameters and resources across the various level of detail. The full-system models are part of the early design stage as well as the architectural level of existing systems. Hence, we distinguish between simulation-based and hardware-metric-based models. The models present full-system approaches with the focus on power consumption.

***Simulation-based Full-system Model***

In [MKO et al. 2002], an instruction set simulator of an embedded system is proposed. The authors developed a model on the architectural level, which takes hardware costs, power, and performance into account. The models build up hardware components, such as the processor core, memory, and cache. They focus on embedded hardware/software co-design system, including the internal bus architecture and pipelines. The application-specific hardware contains a selection of functional modules, which can be either a master or slave. This granularity is not adequate for complex server models, because of high computational and design effort. Furthermore, the system specification does not define the instructions and operations.

The authors of [DAH et al. 2007] develop a full-system simulator considering power, performance, timing, and functional data for the early design phase. They propose a method of predicting the power consumption for various micro-architectural structures using FPGA[126]-accelerated simulation technologies (FAST). The simulator uses a functional and timing model (FM / TM)[127] separately, each executed in an efficient unit designed for these purposes. The functional model streams the instructions to the timing model, which sends the feedback about the power, and performance back to the questionnaire. The authors found dependencies between floating-point units, shift operands, and pipeline stages. Furthermore, the timing models [WJ 1996, DAH et al. 2007] are an alternative approach usually used in the chip and system domain for micro-architectural description. Timing data is a sustained part in the case of software level models. An application and its behavior, for instance, the parallelism of the software[128] and hardware[129], can be modeled using time data values. In addition, the model also considers the timing aspects of the system. The significant factor of management techniques and optimization strategies is the time and related constraints. In [KSH et al. 2009],

[126] FPGA: Field programmable gate array

[127] FM / TM: functional behavior "what", timing behavior "when"

[128] Software parallelism: complex structures and parallel executions, algorithm, compiler options, programming style, independent processing elements, task, or data at the same time

[129] Hardware parallelism: simultaneously instructions, pipelining, machine architecture, hardware multiplicity, superscalar, vector processors, or multiprocessors (shared, distributed memory)

a timing-dependent model is proposed to estimate the dynamic power, which considers coupling impacts. If a coupled effect occurs, a relative switching-based time delay is observable and hence has extra power consumption.

The work of [GFN et al. 2006] proposes a hierarchical model in which every component is again a set of components. Each component has static and dynamic properties whereby the static factors $sp$ are constant and the dynamic properties $dp$ change during runtime, such as the voltage and frequency. The authors present the power consumption model of the server $s$ where $SC$ is the set of components and $N$ the amount of components. They distinguish within the equation in the correlation function $F_{SC}$ between the components and the power function $F_{1\ldots N}$ of each component $c_1, \ldots, c_N$. The developed power model, shown in Equation (3.46), does not simply add the power consumption of the components. It depends upon the correlation between components as well as their static and dynamic parameters. They found that the number of used processor cores has a higher impact in the case of high utilization in comparison to idle or low-level utilization.

$$F(\overrightarrow{s_{sp}}, \overrightarrow{s_{dp}}, s) = \begin{cases} F_{SC} = \emptyset(\overrightarrow{s_{sp}}, \overrightarrow{s_{dp}}) & if\ SC = \emptyset \\ F_{SC} = \left(\overrightarrow{s_{sp}}, \overrightarrow{s_{dp}}, F_1\left(\overrightarrow{c_{sp}^1}, \overrightarrow{c_{dp}^1}, c^1\right), \ldots, F_N\left(\overrightarrow{c_{sp}^N}, \overrightarrow{c_{dp}^N}, c^N\right)\right) & else \end{cases} \tag{3.46}$$

Another system-level simulation tool developed by [BLR et al. 2005], reduces the computational effort to optimizing the accuracy and efficiency, in comparison to functional simulation. They offer a framework that includes alternative heterogeneous component power models to find sufficient power models for an embedded system[130]. Their method combines multiple model types, such as cycle-accurate functional, transaction-level, and instruction-level models dependent upon the accuracy and efficiency constraints of each component. They developed a cycle-accurate functional and behavioral model.

A generalized simulation-based method for power-managed systems is proposed by [BHS 1998]. Benini considers the components in an abstract manner and calls them resources. The set of resources and a power manager builds the system. The power manager "…, translates environmental stimuli into requests to system resources to change their power states, [BHS 1998]." It also includes the power management policies, which are algorithms that try to reduce the power consumption of the components. The authors abstract the external environment (requests) and present it as a component's state change initiated by the power manager. They found that the utilization levels continuously switch between high, low, as well as idle and thus, the power and performance change immediately. The power-manageable resources have multiple power states. This leads to a model that considers the power consumption of inner resource states, the related performance values, and the model includes the switching activity times.

[130] Embedded systems: they concentrate on system on a chip

***Hardware-metric-based Full-system Model***

In [ERK 2006], the authors introduce the *Mantis* method to create a full-system power model and predict real-time power. The authors concentrate on blade servers, but also evaluate the approach on a high-end server. Their method requires real existing hardware, because they calibrate the system to get the power consumption characteristics. In their method, they measure the active current component-level power, including the operating system utilization metrics and performance counters. "*Mantis* estimates total power consumption using a set of user-level system utilization metrics, [ERK 2006]". The authors develop a server power model, including various utilizations for the processor, the memory, hard disk, and network, as stated in Equation (3.47). The coefficients $K_0, K_1, K_2, K_3,$ and $K_4$are server-specific characteristics, such as design and architecture properties.

$$P_{server} = K_0 + K_1 * u_{processor} \mp K_2 * u_{memory} + K_3 * u_{disk} + K_4 * u_{network} \qquad (3.47)$$

The authors found that the transaction-related components use approximately 20 up to 40 percent of the entire power consumption. Furthermore, they state that the idle power is huge on the Itanium server in comparison to the blade. The two highest power consumers are the processor (about $50\%$) and the memory (about $30\%$). The processor power dominance of the entire consumption decreases because of enabled power management strategies. The authors do not characterize the power proportion referring to the utilization levels or component-bounded workloads. The other power contributors are miscellaneous components, such as the disk, net, or fans, which amount to 20 percent proportion. Furthermore, they do not consider the processors' variations and their related BIOS/UEFI settings. The authors measured the system components in isolation and do not consider system-specific effects.

In the further research, the authors of [Riv 2008] introduce five models that estimate the power consumption of a CoolSort machine, a laptop, a Xeon, and an Itanium server system. These machines run a wide range of benchmarks, such as *SPECint*, *SPECfp*, or *stream* to present realistic workloads. The models consist of the coefficients $K_0, K_1, K_2, K_3, K_4, K_5, K_6$ , the relative processor and disk utilization $u_{processor}$ and $u_{disk}$, and the performance counters $P_m,\ P_i,\ P_c,\ P_u$ corresponds to the number of memory bus transactions, the number of instructions retired, the unhalted clock cycles, and the number of last-level cache references. The authors state a constant power model for each machine, as shown in (3.48). The linear model in (3.49) considers the processor utilization. The authors enhance this model, including an empirical factor $F$, as shown in (3.50). They also consider the disk utilization with the linear model based on Equation (3.49). Finally, the performance counters $P_m,\ P_i,\ P_c,\ P_u$ extend the processor and disk utilization-based model, as stated in (3.52).

1. Constant model

$$P = K_0 \tag{3.48}$$

2. Linear processor utilization-based model

$$P = K_0 \mp K_1 * \left\{\frac{u_{processor}}{\max(u_{processor})}\right\}^1 \tag{3.49}$$

3. Empirical processor utilization-based model

$$P = K_0 \mp K_1 * \left\{\frac{u_{processor}}{\max(u_{processor})}\right\}^1 \mp K_1 * \left\{\frac{u_{processor}}{\max(u_{processor})}\right\}^F \tag{3.50}$$

4. Linear processor and disk utilization-based model

$$P = K_0 \mp K_1 * \left\{\frac{u_{processor}}{\max(u_{processor})}\right\}^1 \mp K_2 * \frac{u_{disk}}{\max(u_{disk})} \tag{3.51}$$

5. Performance-counter-based model

$$P = K_0 \mp K_1 * \left\{\frac{u_{processor}}{\max(u_{processor})}\right\}^1 \mp K_2 * \frac{u_{disk}}{\max(u_{disk})} \mp K_3 * \frac{P_m}{\max(P_m)} \mp K_4 * \frac{P_i}{\max(P_i)} \mp K_5 * \frac{P_c}{\max(P_c)} \mp K_6 * \frac{P_u}{\max(P_u)} \tag{3.52}$$

The authors focus on a fixed system configuration, but beneficially for various workloads. They take the memory utilization as performance counters into account, which is a white-box approach including instructions and transitions. A brief overview about the utilization proportion[131] of the processors, memories, and disks is given in the literature of [Riv 2008] for six different benchmarks.

The authors of [RRK 2008] present a constant model that uses *Mantis* [ERK 2006] to predict the full-system power consumption. The operating system reports the performance metrics at each time, which is the basis for the power consumption of the components. The empirical characteristics in the equations have been created by calibration schemes that stress the individual components and measure the performance and power values at the same time. The power model in Equation (3.53) consists of a constant idle power $K_0$ based on vendor specifications, the processor power in relation to its utilization level $u_{processor}$, and the fitting parameter $r$, defined by [FWB 2007].

[131] Utilization proportion: relative statement, such as very high, high, medium-high, medium, medium-low, or very low

$$P_{prediction} = K_0 + (K_1 * u_{processor} + K_2 * u^r_{processor}) \tag{3.53}$$

They also consider the linear regression of the disk utilization $u_{disk}$ by the number of input and output requests or transfers. Furthermore, the model uses the maximal available performance counters $p_i$ involving instruction-level information for model simplification and low overhead. The last model is stated in (3.54), whereby $K_i$ is the coefficient for the corresponding performance counter [Riv 2008].

$$P_{prediction} = K_0 + \left(K_1 * u_{processor} + K_2 * u_{disk}\right) + \sum(K_i * p_i) \tag{3.54}$$

The authors evaluate their approach on various components and characteristics (processor, memory) using four benchmarks. They found that the "… resource utilization metrics correlates to power consumption, [RRK 2008]." The performance-counter-based models are most accurate for each benchmark in the possible configurations. The authors conclude that a linear processor utilization-based model is suitable for an average error smaller than 10 percent. A calibration workload, proposed in [Riv 2008], varies the utilization level of each power-dominant component (processor, memory, and disk) to get isolated values. The authors found that components utilization also differs within the same workload of the calibration suite, which leads to the conclusion that not all components are stressed simultaneously. For instance, the processor utilization is non-linear to the disk utilization because the processor throttles[132] automatically for fewer disk accesses. The processor's power consumption depends also on the various utilization types, such as lower instruction-level parallelism, the number of used (active) cores, or the operations. In [KJC et al. 2014], the authors sum up the power consumption of each isolated part, whereby the miscellaneous power is constant.

The approach of [YSY et al. 2011] is a system-level online power estimation, which uses the on-chip bus performance monitors to capture component activities. The authors focus on the system-on-chip architectures and use an energy state machine (ESM), which distinguishes between static and dynamic energy consumed by each instruction, cycle, or transition. The processor model consists of the static power, the power-relevant performance counters, and their coefficients. The memory model uses cycle-accurate instructions. This approach works for online power measurements at a given cyber-physical system.

The authors of [BJ 2007] use five major components, namely the processor, chipset, memory[133], input/output[134], and disk to model the system. They use performance counters to characterize the component's power consumption of eleven workloads. They found that the processor power relies on the processor-bounded workload and decreases for memory-bounded workloads because a lower number of simultaneous threads are enough to handle

[132] Throttle: clock gating, shut down used units, frequency and voltage scaling
[133] Memory: subsystem includes the memory controller and DRAM power
[134] Input/output: includes PCI buses and attached devices

the job. The maximum of the memory power occurs in the case of read or write transactions. Additionally, the authors found that the processor, which is not considered in the model, does not start some of the component's activities. Some years later, the authors added the graphics processing unit (GPU) to their approach and validated it on a server and a desktop [BJ 2012].

The previous research paper focuses on performance counters or OS-reported utilization levels. The power consumption depends on the systems software, such as the operating system or the type of application executed on the server. The operating system ensures that all jobs will execute. The system schedules and handles a range of jobs and threads, whether or not another management strategy influences the system. For instance, the operating system autonomously allocates and manages memory, controls peripherals, or changes the settings of the frequency to reduce the power consumption.

The software running on the system affects the system power consumption as well. In fact, the software includes various algorithms, multiple coding styles, or compilation optimizations and thus other physical parts of the processors or memories are used. The compiler generates code and transforms data to reduce resource usage or exploit the advantages of the processor architecture. For this reason, the software consumes different power. The workload of the system depends on the input data of the system. It changes dynamically and thus the resources are utilized, but cannot be predicted.

The software-level power models or routine-level OS power models abstract the specific performance counters and use the routines as well as the corresponding time to estimate the power and energy consumption. The authors of [LJ 2003] investigate the use of instructions per cycle, which indicates circuit switching activities and finally the power consumption. They found that the data path and pipeline OS routines consume 50 percent of the overall power. The authors stated that the power consumption depends on interrupts, processes, inter-process controls, or file systems. They present a linear regression model with the parameters $k_1$and $k_0$, shown in Equation (3.55), and include the instruction per cycle. The authors extend the power model towards an energy model by including the time, shown in (3.56), whereby the power and time are specific to the $i_{th}$ OS routine call. This approach works only if we establish the average power of the OS routines. The authors conclude that the detailed software's data, such as OS routines, ensure not absolutely the accuracy of the full-system power model.

$$P = k_1 * IPC + k_0 \tag{3.55}$$

$$E_{OS} = \sum_i P_{OS_{routine},i} * T_{OS_{routine},i} \tag{3.56}$$

In [TRJ 2005], the authors present an energy macro modeling approach for embedded operating systems. They use the white-box approach to find the power-relevant components while observing the OS routines. On the other hand, they measure the power consumption in isolation from the OS energy, as a black-box approach, but consider the customer-level software. Another approach uses the power consumption in the process level [TSW 2009]. The authors create an energy model that reflects the power consumption of a resource with the corresponding utilization level, the related energy consumed by process interactions, and the required time interval. Furthermore, they divide the energy of the components within the system as a function of its states and transitions. Equation (3.57) shows the generic definition of the components' energy consumption, including the power $P_i$ for a time period $t_i$, using a particular frequency and the several transitions $n_k$ consuming the energy $E_k$. This abstract model is refined for hard disk drives, memory, processor, and network.

$$E_{component} = \sum_i P_i * t_i + \sum_k n_k * E_k \tag{3.57}$$

The classic approaches consider particular systems and components to analyze the power management, the energy efficiency, or the optimization. The performance counters of the systems (processor, memory, etc.) are hardware-dependent and therefore, only valid for an explicit configuration or platform on the micro-architectural level. On the other hand, the performance counters show the hardware and software system states, including the operating system. Many research papers [GHD et al. 2009, Han 2007, KJC et al. 2014] evaluate their concrete models for a given configuration considering dynamic power management attributes such as voltage and frequency scaling. These models are analytical real-time models and highly accurate. The approaches do not take the relationships, the dependencies, or the structures between hardware families, generations, or series into account. The portability and comparability are not achieved. Furthermore, the variable selections of workloads and software settings are crucial factors of the power dissipation of a system.

On the other hand, system-based, black-box models try to abstract the hardware details to guarantee the portability. In [Bel 2000], they present a high-level processor power model called *Joule Watcher*, which includes the linear relation between power consumption and performance counters. Those models using performance counters support the real-time power characterization on the fly. The hardware selection limits the number of performance counters and their readings. Usually, all full-system power models use real and direct hardware measurements, as proposed in [ERK 2006, Riv 2008]. The approaches are accurate (about $5 - 10\%$) and fast in comparison to the micro-architectural level, but on the contrary, slow in comparison with real hardware. The model works for existing systems as well as for future systems, which is a benefit of simulation-based approaches. Some research papers use OS-reported utilization levels instead of performance counters to abstract it from the specific hardware, which is also adequate.

Bellosa [Bel 2001] states that most simulation-based approaches do not consider the device’s activate and reactivate time, latency, and performance impacts. The authors of [Bel 2001] argue that an OS-based model and related measurements are not adequate to estimate power consumption because diverse functional units may be used. Additionally, they state that the resource principals differed between internal and external requests.

Another power-related effect is described in [RRK 2008], whereby the models are adequate for processor-intensive benchmarks because the processor power dominates the entire consumption [Ben 2010]. In comparison, the same model does not work accurately for memory-intensive workloads. It further overestimates the power dissipation of the full system. Additionally, the dominance of the processor’s power consumption becomes relative when the processor uses management technologies, such as voltage and frequency scaling. In such a case, the proportions of the memory power contribute to the entire consumption increases and become more relevant. In addition, the authors present a simple constant power model in [RRK 2008], which is very inaccurate because of the model’s simplicity. “All these models have in common that either they are too simple and inaccurate or very specific and complex, [GFN et al. 2006].”

In the early design phase, the applications or architectural decisions are uncertain. The level of detail varies between chips, components, or the system so that the designer cannot estimate accurate power consumption [DAH et al. 2007]. On the other hand, the physical level approaches are not usable at the system or component domain because of their detailed granularity and time aspects for the simulation.

Benini et al. [BHS 1998] provide a generic approach that abstracts the inner system components from the external requests. A power manager schedules the state changes with respect to management policies. The authors do not focus on the resource power models themselves, but provide a simple full-system model on the gray-box level. Their method considers multiple inner-component states, but do not concentrate on various component families and generations. Furthermore, Benini et al. do not investigate into the various workload types and their characterizations.

### 3.4.3 Environment – Server System Domain

The work of [FWB 2007] shows the various power distributions at various hierarchies within the data center. The rack server power is typically connected to the power distribution unit, which is part of the facility or data center. The power management strategy can rely on the data center specification for a single server system or a complete 19-inch rack enclosure. A configured power limit at the server’s firmware is only valid for the server-specific power management. It is another interface, but influences the same server settings. A peak power value is the maximal reached limit for the server. The server manages the power consumption autonomously via dynamic voltage and frequency scaling, which is an interior contractual enforcement because of energy efficiency. On the other hand, the firmware controls the physical limit. For instance, the server firmware can set the maximally available power state of the processor, which is independent of the operating system. It avoids overloading of electrical circuits. The firmware or the processor directly enables the maximal speed of the fans to start cooling. The server environments, such as increasing utilization levels and more executed jobs are the reason why the system consumes more power. This is part of the data center's virtualization, scheduling, and allocation techniques. The data center manager provides power for multiple systems that share the power infrastructure. The characteristics of the system utilization are not known or predictable. On the other hand, the customer does not care about the infrastructure.

The authors of [BC 2010] propose a power model for a virtualized cloud. The coefficient $K_0$ is the offset at the idle state. The other coefficients weight the performance counters $p$ of the components. The authors determine the linear relationship between the component's utilization and the total power consumption $P_{total}$.

$$P_{total} = K_0 + K_1 * p_{processor} + K_2 * p_{cache} + K_3 * p_{DRAM} + K_4 * p_{disk} \tag{3.58}$$

They also find that the accuracy depends on the interrelation between the input variables. Therefore, they enhance their model, including the correlation between the components, as shown in the following equations. They classify the workload and hence the model into processor-bounded and I/O-bounded processes. The authors proposed two linear regression models with correlated system events with the coefficient $a_1, a_4$ as offsets to the idle state. The other coefficients are again weight factors. They also investigate the baseline and dynamic power for the virtual machines.

$$P_{\{processor,cache\}} = a_1 + a_2 * p_{processor} + a_3 * p_{cache} \tag{3.59}$$

$$P_{\{DRAM,disk\}} = a_4 + a_5 * p_{DRAM} + a_6 * p_{disk} \tag{3.60}$$

$$P_{total} = \alpha * P_{\{processor,cache\}} + \beta * P_{\{DRAM,disk\}} \tag{3.61}$$

In addition, various workload scenarios utilize the system in several ways, and the system settings play an important role. The operating system influences the power consumption [FWB 2007, RRK 2008]. The application software uses various circuits with respect to the optimization settings [YVK et al. 2000, LJ 2003] and changes the power consumption as well. Therefore, it is necessary to examine diverse server systems and to observe the power consumption to change the coefficients.

The utilization level of the physical domain focuses on instructions and operations on an architectural level. The benchmark suites utilize the components and the systems in several ways. At data center domain, virtualized environments, scheduled jobs, or applications adjust the utilization levels. The utilization highly depends upon the performed task in the server system domain. In addition, the data center operator perhaps does not know about the explicit application or resource utilizations.

This section does not include further literature on power analysis in the physical system domain because this thesis does not consider circuit-based models. The white-box section describes a brief overview of the low-level models, including instructions and activities using the physical and logical description of the resources. The details are more granular for white-box models in comparison to the black-box models. The resource abstraction increases because the facts, functionality, activities, or instructions are unknown. At the same time, the complexity of the resources decreases because interdependencies or relationships are considered. Gray-box models use state machines to estimate the power consumption of the components and the system, whereby the activities or instructions are defined in an abstract manner. Gray-box approaches rely on conceptual definitions; and on the other side, black-box models use contextual or external description. In fact, the researchers work continuously on power models of components at various level of detail, as shown in Figure 35. The academic research concentrates on physical and component-based model for power and energy aspects. The models are specific for a given software or hardware configuration. It is necessary to investigate the coefficients and weight for each setup.

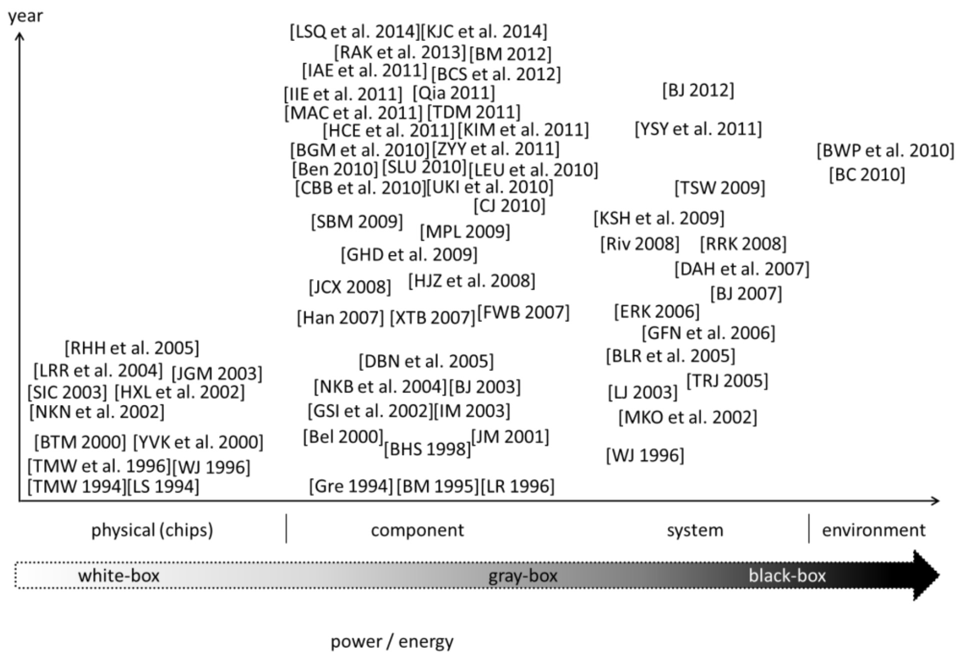


Figure 35: Power and energy models in recent years

## 3.5 Utilization Algorithms and Approaches

The power-based models work with performance counters that characterize the component's power. Another aspect is the utilization of the server system domain whereby the performance counters represent the chips, components, or system activity in another way. If the utilization level of a component increases, the value of the performance counters will also grow. The utilization level indirectly affects the other aspect-based models. In fact, the resource utilization level is part of the power/energy or performance model inputs, which is sometimes depicted alternatively because of the level of detail within the domain. Therefore, the utilization is a concealed aspect and not a separate model. The utilization levels are the key elements for the workload definition, which builds up a realistic scenario. Section 5.3.2.2 defines the workload characterization.

## 3.6 Thermal Algorithms and Approaches

The resources utilization results not only in power consumption, but also in leads to thermal dissipation. The total energy consumption includes the heating and cooling aspects. Permanently running fan motors or air-conditioning systems guarantee the temperatures of operation as well as humidity conditions. The thermal behavior relies on the operating condition of a system and on resources, chips, and their related states. A heat transfer changes the inner thermal state. The thermal energy considers the volume flow[135], density, and specific heat. Fourier's law of heat conduction also describes the thermal behavior heat flux using the heat flow, thermal conductivity, and thermal gradient. Additionally, Newton's law of cooling includes the heat transfer rate and the thermal resistance. Therefore, the material properties influence the temperature distribution of the resource [FOG 2008]. The corresponding management techniques differ because of the reference quantity. For example, when a constant volume flow and steady fan speed must be kept, the algorithm can change the air pressure or density by a given temperature difference. On the other hand, when the pressure is constant, the volume flow depends upon the speed and pressure. The volume flow changes proportionally to the speed, the pressure depends upon a square form of the speed, and the power consumption is cubic to the speed [Jun 1999].

The approach proposed in [SAS 2002] focuses on thermal resistances and capacitance (RC) on integrated circuits at the physical levels. The authors analyze the structure and architecture of the die, heat spreader, and heat sink, which form the base of the electrical functional blocks. The models in [SAS 2002, PZH et al. 2005, YS 2005] estimate the temperature of the various micro-architectural units within a chip and the cooling package. The *HotSpot* method defined by [SSS et al. 2004] is also a thermal RC-based model. Within the micro-architectural level, an RC circuit describes each chip's heat (power) dissipation. The authors use the heat spreader

[135] Volume flow: depends upon the cross section, the volume speed/velocity, and the flow types (swirl, linear, laminar, or turbulent)

and sink layers to add further RC circuits to support the package characteristics[136] and convection. The authors of [KS 2005] extended the *HotSpot* method to estimate the power from the physical hardware resources. Therefore, they add hardware monitoring events, such as performance counters, and estimate the temperature at the floor plan level. The authors of [AC 2003] characterize the thermal behavior of a low-profile, low-cost power package. Another adjustment of the *HotSpot* approach is done in [MNR 2007]. The authors expand the model, including various layers about the silicon-on-insulator technology, chip interconnect, or packaging. Another thermal characterization of packaged semiconductor devices is done in [SXC et al. 2000]. These approaches are very complex and need much time to create an electric model based upon the functional and architectural levels. Furthermore, they require multiple measurements and iterations to analyze the diverse applications.

A physically based model is introduced in [BBT et al. 2014], which generates a thermal map along the chip or the package. The authors use a gray-box identification to determine the coefficients of the material[137] and heat conduction of the processor. The gray-box approach is intended to find the characteristics of the discrete-time, discrete-space model, which the authors proposed in the paper. Furthermore, they differentiate between the copper and the silicon layer because of their structure and spatial characteristics.

In [SA 2003], research is done not only for RC-based models, but also for the effects of voltage and frequency scaling on thermal behavior. The authors state that the temperature depends upon the quadratic voltage and linear frequency at each frequency. In [BKW et al. 2003, WB 2004, MB 2006], the authors present a thermal model that includes the processor characteristics as well as its heat sink. The thermal model consists of a thermal resistance and capacity, whereby the utilization represented as event counters initiate the energy consumption and finally the temperature. The energy of the processors is the input value for the energy model of the heat sink, as shown in (3.62). The difference of the heat sink energy $\Delta Q$ is calculated by the dissipated processor power $P$ over an elapsed time $\Delta t$. The heat sink stores thermal energy, such as heat, which consists of a constant $c$, the mass of heat sink $m$, and the heat sink's temperature increase $\Delta T$. The thermal energy is in balance and thus, the heat sink's energy output is equal to the heat sink's energy input minus the stored heat. Due to the convection, the energy is formulated by the constant $\alpha$, the thermal resistance $R$, and the ambient temperature $T_0$. This part is transformable into Newton's law of cooling [BKW et al. 2003]. The thermal resistance delivers the processor's heat to the ambient air, which relies on convection. They found that the coefficients about the increasing and decreasing temperature characteristics must be defined separately.

$$\Delta Q = \int_{t_1}^{t_1+\Delta t} P(t)dt = c * m * \Delta T = \frac{\alpha}{R} * (T - T_0) * t \qquad (3.62)$$

[136] Package characteristics: material, physical, or geometrical
[137] Material coefficients: mass density, specific heat, or thermal conductivity

The authors propose an exponential function to cover the real thermal behavior of the processor. The quadratic relationship is shown in Equation (3.63), whereby the coefficients $a_2, a_1$, and $a_0$ are processor-specific. The processor's temperature $T(P)$ has a square and linear regression to the power consumption values of the processor $P$.

$$T(P) = a_2 P^2 + a_1 P + a_0 \tag{3.63}$$

The authors focus on energy-aware scheduling and balancing of tasks with different energy characteristics, but also consider the thermal behavior of various processors. The authors in [Liu 2011, BKW et al. 2003] use the utilization-based thermal models. They assume a linear relationship between the processor utilization $U$ and the temperature $T$, as shown in Equation (3.64), with the coefficient $K_1$. In addition, they also use an additive model for temperature decrease, which is independent of the utilization, as shown in (3.65).

$$dT = (K_1 * U)dt \tag{3.64}$$

$$dT = -K_2(T - T_0)dt \tag{3.65}$$

After combining and solving the differential equation[138], the result is an exponential time-based function similarly to [MB 2006], which focuses on the utilization $U$, the ambient temperature $T_0$, and the coefficients $K_2, K_1, K_0$. The first part of the equation depicts the dynamic thermal behavior of a component, such as temperature increase or decrease, as long as the system does not reach the steady-state temperature. The steady-state temperature is the second part of the equation, which results when the component is at the same state over a period. The offset $T_0$ defines the temperature start condition.

$$T(t) = -\frac{K_0}{K_2} e^{-K_2 t} + \frac{K_1}{K_2} U + T_0 \tag{3.66}$$

The authors found that the temperature behavior over time correlates to various workloads. Thus, they changed their approach toward regular utilization, which is a linear function of the average temperature and vice versa.

In [HCG et al. 2006], the thermal energy produced by a component $Q_{component}$ is calculated by the consumed power $P$ at a specific utilization level and the required time, shown in Equation (3.67). The utilization-based power is the sum of idle power $P_{base}$ and the relative power[139] weighted by the utilization level. A utilization factor of one refers to the maximum consumed power; a utilization factor of zero states an idle case. The heat model of the processor is based upon hardware performance counters because it is non-linear. The authors used the temperature model, as shown in Equation (3.62).

[138] Combining and solving the equation: further details shown in [Liu 2011]
[139] Relative power: between $P_{max}$ and $P_{base}$

$$Q_{component} = P(utilization) * time \tag{3.67}$$

$$P(utilization) = P_{base} + utilization * (P_{max} - P_{base}) \tag{3.68}$$

The approach uses discrete-time steps on an airflow graph, which considers the inter-component heat-flow, the intra-machine airflow, and the inter-machine airflow of the considered system.

The thermal model in [KLL et al. 2008] is a regression-based approach to characterize thermal behavior of various applications. It is a run-time method that uses performance counters and thermal sensors of the processor analyzed in every micro-architectural detail, which is a white-box approach. The authors observed the micro-benchmarks, the corresponding on-die temperatures, and the hardware events. They developed a process-based thermal model, see Equation (3.69), which considers the local and global temperatures $t_{lp}, t_{gp}$ of the process $p$ weighted by $w_{lp}$ and $w_{gp}$. In addition, the authors aggregated the measurements of one or more applications and defined a "weighted average of the local and global-temperature components, [KLL et al. 2008]" as shown in Equation (3.70). The overall local and global temperatures $T_l, T_g$ are weighted by $w_l, w_g$, which come from several processes as average values.

$$T_{process} = w_{lp}t_{lp} + w_{gp}t_{gp} \tag{3.69}$$

$$T_{overall} = w_l T_l + w_g T_g \tag{3.70}$$

The industry trend of shrinking device technology results in higher temperatures and lower reliability. More semiconductor components are closely positioned and the device density is extreme, which results in more performance in the same space. Each resource is defined by several temperature limits or ranges regarding long-term reliability[140], functionality, and damage due to their thermal performance [Mic 2007]. Operating temperature intervals ensure unrestricted functional reliability. Increasing the temperature has a negative impact on working behavior. In reference to the Uptime Institute, an increase of 18 degrees Celsius doubles the equipment and server failure rate [ERK 2006, LU 2009]. Researchers at the Los Alamos National Laboratory and Uptime Institute proposed that it is a continuous process while the temperature increases. Because of the high ambient temperature of semiconductors, the reliability and functionality decrease because huge errors occur concerning permanent silicon damage [Lin 2009]. Hence, the temperature is a dominant factor in performance and reliability.

[140] Reliability: meantime to failure (MTTF), meantime between failure (MTBF), failures in time (FIT), time-dependent dielectric breakdown (TDDB), or negative bias temperature instability (NBTI)

Usually, we design power and performance management such that a) the functionality limit is reached and b) the damage limit will never be reached. The thermal design power (TDP) is a power limit to prevent overheating and fulfill reliability requirements. In fact, the damage limit is the largest allowed device temperature. Figure 36 shows the three thermal limits, which in general are part of every resource. Typically, the power and thermal management techniques work between the functionality and damage limit because the time is much longer compared with the time between the reliability and functionality limit. The power consumption of the fans increases with the growing thermal conditions, and thus the working efficiency suffers as a result. In this thesis, we consider thermal limits with the focus on the thermal hotspots within the system.

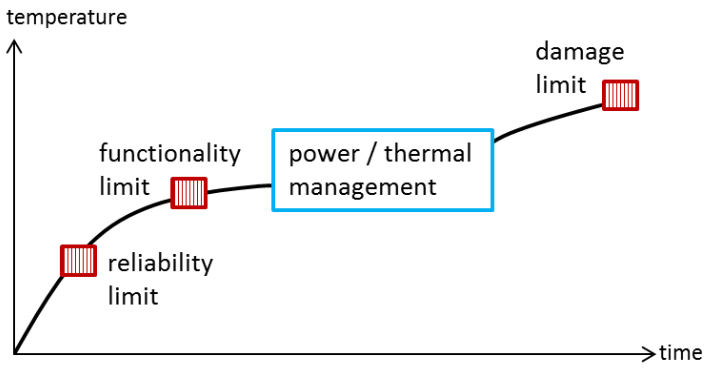


Figure 36: Thermal limits of the resource [Ste 2012]

These three thermal limits are resource-specific. Table 17 shows the well-known limits and the related processor signal names [Don 2006]. The fan control algorithms use the thermal trigger point TCONTROL as the offset point. The PROCHOT signal indicates the processors' on-die temperature and activates the throttling mechanism by either using a less consuming throttle state or setting a lower voltage or frequency. "PROCHOT is a fixed temperature threshold calibrated to trip at the max specified junction temperature, [RHA et al. 2007]". The most critical thermal limit is the THERMTRIP signal, because the system shuts down to avoid circuit damages. "THERMTRIP is a catastrophic shut-down event, both on the CPU and for the platform. It identifies thermal runaway in case of cooling system malfunction and turns off the CPU and platform voltages, preventing meltdown and permanent damage, [RHA et al. 2007]".

Table 17: Processor thermal limits [Don 2006]

| Temperature | Limit | Processor signal name | |
|---|---|---|---|
| **Low** | Reliability | TCONTROL | Thermal trigger point |
| **Medium** | Functionality | PROCHOT | Processor hot |
| **High** | Damage | THERMTRIP | Thermal trip |

In [RHA et al. 2007], the authors state four reasons why the temperature measurements, such as parameter variance or manufacturing thermal control are inaccurate. They cover the temperature measurement on the die considering the hottest location and the diode distance. The digital thermal sensors (DTS) support a higher accuracy in comparison to the analog thermal diodes and read the temperatures for each core separately. Additionally, the DTS offer the opportunity for a graceful shutdown because a notification to the operating system occurs before the THERMTRIP happens.

In [Han 2007], the authors developed a thermal model of the processor which uses the scalar coefficient $\tau$, the p-state-based power $P_{p-state}$, and the ambient temperature $T_{ambient}$. The authors proposed a fixed value of the coefficient $\tau = 1.25$. This is a simple equation (3.71) to model the thermal behavior of the processor. The deviation of over- and under-estimation is approximately 4°C and highly accurate in comparison to the micro-architectural approaches with less model effort.

$$T_{processor} = \tau * P_{p-state} + T_{ambient} \tag{3.71}$$

The authors in [HS 2007, Han 2007, HKG et al. 2007] found that the processor's temperature fluctuates within 50ms but the ambient temperature ramps up in minutes. They found a feedback loop: the thermal increase requires a speed-up of the fans, which consumes more power. The higher power dissipation leads to a larger cooling need and thus, the fan speed increases as long as it does not reach the maximum rotation speed. The authors state that the ambient temperature is only an offset of the measured processor temperature. The ambient temperature increase results direct proportionally to the processor temperature increase in the case of constant fan speed.

In comparison with the processor-based thermal models, the memory models concentrate on the hardware architecture of the modules. The approaches split the printed circuit boards (PCBs) into chip-based areas on which the model is based. The authors of [LZZ et al. 2007] developed a thermal model for single fully buffered DIMM without any thermal interaction to other modules, but on the other side consider the interaction between the buffer and memory chips. They state that the memory throughput determines the heat generation; a higher airflow velocity leads to faster heat dissipation; a higher ambient temperature produces a higher module offset; and finally the type of the heat spreader influences the heat creation. Equation (3.72) shows the additives model of the buffer temperature $T_{buffer}$, where the thermal resistances $R$ are from the buffer to the ambient environment and from the DRAM to the buffer. The power of the buffer $P_{buffer}$ and the DRAM $P_{DRAM}$ are multiplied with the resistances, and the ambient temperature $T_A$ is the offset. The DRAM temperature is calculated in the same manner, but using the thermal resistances $R$ of the buffer to the DRAM and from the DRAM chip to the ambient, as shown in Equation (3.73).

$$T_{buffer} = T_A + P_{buffer}R_{buffer} + P_{DRAM}R_{DRAM\_buffer} \quad (3.72)$$

$$T_{DRAM} = T_A + P_{buffer}R_{buffer\_DRAM} + P_{DRAM}R_{DRAM} \quad (3.73)$$

The authors found that the temperature of the buffer is much higher in comparison to the DRAM chips. In [QXY 2008], the authors state that the thermal characterization of memory modules does not only rely on self-heating, spreading heating, or adjacent heating resistances, used by [LZZ et al. 2007]. They also involve a computational fluid dynamics (CFD) model, which represents the thermal behavior under changing airflow. In addition, the authors used the largest device operating (junction) temperature from the supplier to estimate the memory cooling capability. The authors analyze the variation of memory heat spreaders[141] and their influence on airflow, which depends upon the system operating curve, the fan characteristics, and the system layout. In [SBA et al. 2011], the thermal model focuses on the single-sided, mounted memory module and distinguishes between the chips assembled in or against airflow direction. The authors develop an RC-based model that estimates the thermal behavior of each chip under consideration of optional buffers. They found that the chip temperatures vary by over $10°C$.

The approaches are based upon resistance, but do not consider the thermal interaction between resources from the same type, from other types, or inner dependencies because of the circuit structure as well as the architecture. Furthermore, the system-wide thermal effects are not taken into account. The processor models only consider local hotspots, whereby the temperature is non-linear to the utilization level and the input voltage, but linear to the frequency. The temperature changes slowly in comparison to power.

The thermal and flow characterization of a blade system was done in [Erd 2013]. The authors develop a single transient black-box server model, which considers the airflow rate, pressure, time constants, thermal conductance, and capacitance of the system. Finally, they introduce a black-box model that uses the thermal capacitance of the server, which reports the temperature difference between the inlet and outlet from the system. A first-order differential equation describes the time-dependent outlet air temperature and stream. This approach was investigated experimentally under a constant fan speed and executing a rack shutdown. The authors found that the capacitance depends on the servers' mass and specific heat because of the volume and material.

In the system and the environment (data center) domain, the common thermal approach is based upon computational fluid dynamics simulation. This method provides a 3-dimensional, temperature-based view of the entire system. The authors in [CKS et al. 2007] propose *ThermoStat*, a statistical thermal modeling tool for the server and racks. They considered the system's geometry, the materials, the components' placements and their power dissipation,

[141] Heat spreader: the adequate thickness, structure, tolerance (air gaps), or mechanical properties

and the cooling mechanism, including the air characteristics and thermal properties. They focus on the correct component placements within a server as well as the server's location within a rack. Therefore, they simulated the fan failures with regard to the inlet air temperature.

In [ISS et al. 2012], the authors focus on the servers' thermal mass[142] and response at fixed power levels using diverse fan speeds, which corresponds to the various airflow rates. They state that the heat-transfer coefficient increases when the airflow rate rises. The thermal total energy depends linearly on the exact heat of air, the temperature difference across the server, the thermal mass, and the server temperature under time conditions. The authors state an additive weighted temperature regression model of the server, as shown in Equation (3.74), whereby the coefficients $K_{HDD}, K_{RAM}, K_{PS}, K_{HS}, K_{chassis}$ and the temperatures $T_{HDD}, T_{RAM}, T_{PS}, T_{HS}, T_{chassis}$ are specific to the hard disk, memory, power supply, heat sink, and chassis.

$$T_{server} = K_{HDD}T_{HDD} + K_{RAM}T_{RAM} + K_{PS}T_{PS} + K_{HS}T_{HS} + K_{chassis}T_{chassis} \tag{3.74}$$

The authors found that the chassis has the largest influence on the server temperature and thermal mass, followed by the hard disks, power supply, processor, and memory modules. They extract the thermal mass and conductance information, which are included in the CFD simulations. A couple of research papers [ASS et al. 2014, PV 2014] focus on the servers' thermal mass or capacity for CFD simulations. The authors in [Qih 2008] couple the heat and mass flow of the entire system. They proposed a transient multi-scale thermal model, which was proofed by an isolated gate bipolar transistor module integrated within a server enclosure.

The work done in [BBB 2011] focuses on the steady-state thermal model of a multi-core processor by linking the cores' power and their temperatures. The authors developed a linear regression model that collects the power and temperature measurements of the processor. They assume that all cores are within the same small neighborhood and define a thermal transient time-discrete model with the following equation (3.75), whereby $n$ determines the time index, $T$ is the temperature, and $P$ the power vectors with the dimension of cores $C$. The square matrices $A, B$ are calculated from the cross-product of $C$. The authors assume that the processor is in steady state and therefore $T[n+1] = T[n] = T$. The temperature depends on the ambience as well as the core temperature, and the model distinguishes between the working and sleep states.

$$T[n+1] = A * T[n] + B * P[n] \tag{3.75}$$

[142] Thermal mass: the capacity to store the heat [IBS et al. 2012]

The authors stated that the power consumption of a component grows when the core of a processor is active, because of the shared clock and power domain of the processor. The base power exists if the processor core is in operation and corresponds to the lowest of all the frequency values of the cores.

Furthermore, the entire processor power depends on the number of active cores, the power of the fully busy cores, and the system power. They measured the temperatures and power consumption based upon core-based activation patterns of various operating frequencies. They found that "the configurations with the same number of active cores have similar power consumption levels, [BBB 2011]," the power consumption is symmetric to the number of cores but non-linear to the frequency.

A numerical data center model is developed in [IGB et al. 2010], whereby the power and airflow of the server vary over time. The authors focus on transient CFD modeling. They found that if only the power varies, the buoyancy effects are the main factors for inlet temperature variation. The CRAC airflow changes are directly observable via the rack inlet flow during the time. The inlet temperature changes faster than the power when both are flexible, because of on-top or edge recirculation. The server power does not influence the inlet temperature when a constant airflow is established. In addition, the same authors propose a transient model of data centers in [IBS et al. 2012], which considers the airflow changes. They also model the server thermal mass that is beneficial to the transient analysis. The authors state that the power dissipation varies over the time and depends on the fan speed inside and the pressure across the servers. They analyzed the rack power under constant and variable CRAC airflow. Furthermore, "the server was modeled as a simple thin plate, [IBS et al. 2012]" which has a thermal mass considering the material mix of steel and copper. The specific heat capacity and conductivity are part of the thermal mass model and therefore, part of the thermal boundary conditions, which is a time-dependent factor to read the steady state. The authors found that the airflow variation has a more rapid result of the inlet temperature in comparison to power changes.

The thermal model in [BWP et al. 2010] uses the heat dissipation and power consumption of the cooling systems within the data center. The authors' model the fan, the rack, the CRAC, the chiller, and the cooling tower, including the following parameters:

- Fan speed
- Volume flow rate
- Pressure drop
- Mass flow rate of air
- Heat transfer coefficients
- Heat exchanger effectiveness

to represent the heat flow from the rack level to the chiller. Due to the variety of cooling systems within a data center, the authors analyzed the inlet air temperature of the rack systems and the dependencies towards the infrastructure of the data center cooling systems. The authors found that the increase in rack-air inlet temperature of approximately five °C improves the coefficient of performance (COP) by about eight percent. They also state that a temperature rise across the racks is more efficient in comparison to a rack inlet temperature increase. Similarly, the authors in [JVG 2010] propose a data center thermal model that considers the heat interference and the cooling but uses ambient sensors to detect the temperatures. The authors of [RZB et al. 2012] state a model-based approach of the data center's thermal environment. The authors develop a dynamic rack-inlet temperature model.

The simulation-based approaches require real servers to measure the temperatures of each component to find the weight coefficients. This approach is of limited practical utility in investigated time, effort, and presence of constant environmental conditions, such as humidity, temperature, or utilization levels. These tools are useful to analyze particular components. The easy equations use the analogy between electrical and thermal properties with resistance and capacitance. On the contrary, the equations become more complex for CFD simulations, which require an extensive analysis of the real hardware under static and dynamic considerations. The models are more accurate and complicated because of the entire system behavior, such as airflow. Another critical factor for temperature approaches is the simulation speed, because thermal changes are substantially faster in comparison to the total simulation time and therefore, differ across several steps.

In fact, the researchers work continuously on thermal models at various level of detail, as shown in the Figure 37. Within the last recent years, the authors focused on the system and the environment domain. The thermal models, especially the coefficients and weights, need to be adapted for each particular server configuration.

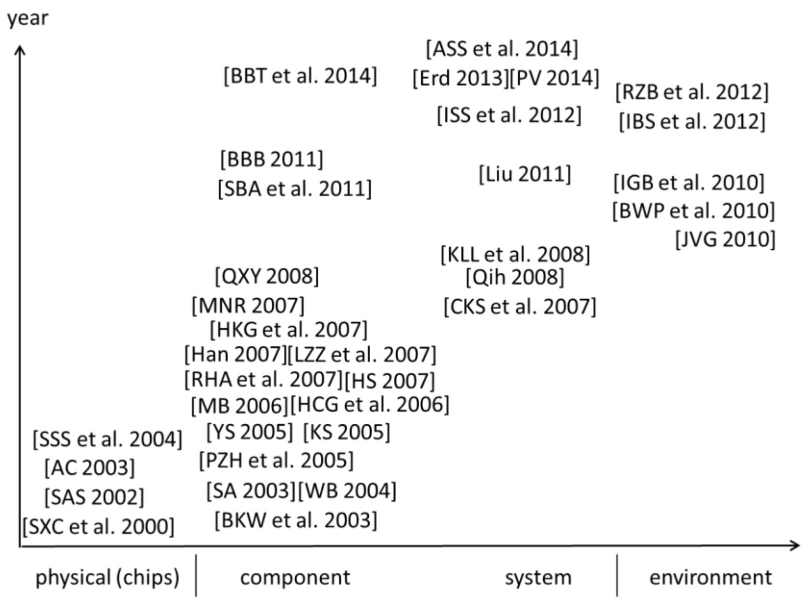


Figure 37: Thermal models in recent years

The cooling costs increase by the growth of thermal dissipation. At a certain level, the cooling costs grow strongly with respect to the dissipated temperature [GBC et al. 2001]. The computational density trend, such as the larger amount of transistors or capacitance and the higher frequencies, are causes of the thermal increase. It is necessary to find the break-even within the thermal-to-cost ratio; otherwise the correct energy efficiency is impossible. Moreover, high temperatures will have some negative effects, such as reduced reliability of the resources. Hence, the temperature is a dominant factor of performance.

## 3.7 Performance Algorithms and Approaches

The system and resource performance indicate how powerful the system operates. The performance models estimate the peak value, the average values, or the performance per watt on the basis of resource characterization. Usually, the system or resource executes a set of workloads, such as benchmarks (suite), and the researcher observes[143] the corresponding performance counters, events, power values, or temperatures. The results of these empirical studies and analyses form the basis for the performance models. Furthermore, the stress findings are relevant criteria for statistical and experimental models validated with various hardware settings. The performance characterization depends on the executed instructions (floating-point, integer) and related activated functional units on the hardware [DEP et al. 2009]. The compiler settings influence the software execution, whereby the operating system changes the device frequencies on demand. The authors of [FM 2002] illustrate the performance levels and time spent within various frequencies at two MPEG[144] scenarios using

[143] Observation: measuring, profiling, tracking, or logging
[144] MPEG: moving picture experts group

the video playback *LongRun* and *Vertigo*. They found that neither the frequencies, nor the time is equal. This indicates that the amount of hardware devices, the types[145], or the architecture requires an adjustment of the model, whether through weighting factors, model design, or diverse descriptions based upon diverse system domains[146].

The performance model in [MKO et al. 2002] is based upon the architectural level. The authors analyze the design and architecture of an embedded system and its elements. They consider the behavioral level, such as using the events that execute instructions of one or more system elements.

The authors in [YZ 2011] estimate the performance of the various memory modules in embedded systems, whereby they focused on cache activities. They completed a compiler static analysis and dynamic profiling through a simulator. The data traces show the realistic accesses. The authors analyze the data objects and classify it into four different *dblocks*[147], which conflict which each other. The authors state a cache conflict graph (CCG) to determine the cache performance criteria, such as the cache misses or hits. Each edge within the graph has a weight that shows the amount of memory accesses. This approach requires a scan of the assembly code to get the *dblocks*, the cache table, and the cache conflict graph to estimate the hits and accesses. [MHS et al. 2009] propose that the main memory latency consists of the L3 cache misses[148], the quick path interconnects (QPI), and the integrated memory controller (IMC) per DIMM latencies, as shown in the following equation. They found that the absolute values rely on the processor's core and uncore frequency.

$$latency_{RAM} = latency_{L3\ MISS} + latency_{QPI} + latency_{IMC/DIMM} \tag{3.76}$$

Additionally, the authors stated that the read performance of a memory module increases with the growth of the data as well as with the amount of the cores. Moreover, the write performance of the L3 cache and RAM increases with higher number of cores. The authors analyzed the performance of L1, L2, L3, and RAM for concurrent read and write accesses. They measured the single-core and multi-core bandwidth accessing the same memory module. The authors found that the bandwidth of L1 and L2 caches is linear to the number of threads, because the threads executing on each core are independent. The RAM and L3 cache performance have only doubled by the quadrupled threads, because the system shares the L3 bandwidth.

[145] Hardware types: server types, processor generation, single inline, or dual in-line memory modules
[146] Descriptions on various system domains: instruction-based, functional blocks, state transition diagrams, or spread sheets
[147] *Dblocks*: "The *dblock* is defined as a contiguous sequence of data within the same data object that is mapped to the same cache set in the data cache, [YZ 2011]".
[148] L3 cache misses: latency of L3 cache misses includes the miss rate

The authors in [ZT 2011] experimentally analyze a memory system that uses the *Intel Nehalem* and *Westmere* micro-architectures via the *triad* benchmark. They stated that the data locality[149] influences the performance of the applications because of the throughput of the integrated memory controllers and their shared resources, such as bandwidth. The authors found that the on-chip memory controller has a better bandwidth with the increasing number of tasks in comparison to the off-chip memory controller. The processor cores request the access to the memory simultaneously. In their configuration, the memory bandwidth is the ratio between accessed data per processor cycles. Herein, the authors define the accessed data by the size of the cache lines multiplied with the last-level cache (LLC) misses at a processing frequency, as shown in Equation (3.77). The *Intel Nehalem* processor has a cache line size of $64bytes$ and a frequency of $2.27GHz$.

$$memory_{bandwidth}\ [MB/s] = \frac{size_{cache\_lines} * LLC_{MISS} * f_{processor}}{processor_{CYCLES}*10^6} \tag{3.77}$$

The system performance depends on the balance between local and remote memory accesses. Furthermore, the authors proposed an overall system throughput model that sums the instructions per cycle (IPC) values of each process $p$.

$$IPC_{total} = \sum_{p\,\in\,processes} IPC_p \tag{3.78}$$

The authors in [KKK et al. 2012] propose a memory behavioral model. The authors found that the memory performance does not only linearly scale to the number of processor cores and the related parallelized software. They create a performance model that uses the number of parallel tasks (iterations), the amount of parallel sections (loops), and the computations without and within locks. Furthermore, the approach works on the number of all instructions and DRAM accesses to model the performance.

Those performance models all constitute the need of an explicit hardware and software configuration that traces and estimates the event counters. The memory architecture, the data locality, and the cache misses are criteria for the performance modeling approaches. The workload type generates performance differences because of parallelization or shared accesses. It is necessary to measure the performance values on a real server system with an existing job to make a general and reliable statement. A forecast or prediction is only possible for an explicit workload, where the read-to-write ratio is established, e.g., for memory modules. The exact performance values of the instructions need to be known. In conclusion, the amount of performance models is less in comparison to the power and thermal models. Usually, the performance is measured on the real hardware and seldomly predicted because of a priori unknown resource claims, the high-required level of detail, and the non-existing workloads. Figure 38 shows the performance approaches mentioned during the recent years.

[149] Data locality: local / remote caches or RAM

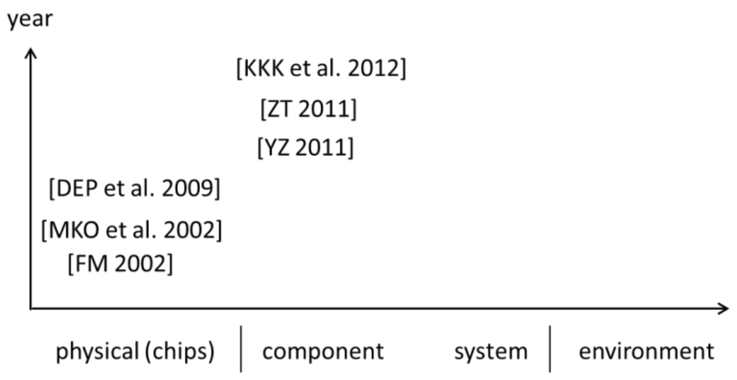


Figure 38: Performance models in recent years

In this thesis, we measure the performance using several benchmarks. We do not cover the performance approaches because we focus upon the power and thermal models. The thesis models are validated via real hardware and realistic software running on the server. The performance criteria will verify the performance assumptions and check the performance monitoring. The performance models help to estimate the energy efficiency of the resources and the systems.

## 3.8 Efficiency Algorithms and Approaches

The efficiency is a criterion to determine the productivity, whether in power, thermal, or performance. The efficiency provides the ability to avoid costs, airflow, humidity, carbon footprint, efforts, energy, time, or money at the considered aspects. The ratio between output and input values is the basis of management techniques and optimizations to ask the questions of how and where to optimize. Table 13 shows some efficiency metrics within the various server domains. The IT or load density is a key factor for power models. For example, the thermal models of the server resources use the capacitance and resistance to predict the temperature. The server performance, such as bandwidth or the instructions per cycle are indicators to estimate the response time or how much time a job consumes. The focus on efficiency such as power usage efficiency, data center cooling system efficiency, IT equipment efficiency, or energy efficiency characterizes the algorithms and approaches. There are fewer independent theoretical approaches. A guiding principle is to create a unique and separate model under particular circumstances and specific conditions. The following equations show the energy as well as power efficiency calculation under thermal constraints, which is the focus of this thesis.

$$energy\ efficiency = \frac{performance}{energy} = \frac{performance}{time * power} \tag{3.79}$$

$$power\ efficiency = \frac{performance}{power} \tag{3.80}$$

The energy efficiency increases when the system executes the identical work, if the job is processed more quickly consuming the same power. Another method is to improve the performance, while the energy consumption is constant. Of course, the aim is to maximize the performance and reduce the energy consumption either via time or/and power at the same time. The efficiency is inversely proportional to the energy or power and directly proportional to the performance. The efficiency is constant when the performance and power simultaneously increase or decrease by the equal factor. Therefore, the aim of energy efficiency optimization relies on a fixed performance demand or a predefined power limit, which lead to power/energy minimization or performance maximization. In addition, an increasing frequency may lead to high temperatures above the functionality limit or the damage limit, which potentially generates more failures. On the other hand, a power reduction to a lower device state leads to a time increase and thus, higher operating costs.

## 3.9 Server System Models and Simulation

Figure 39 summarizes the studied system-based models considering power/energy, thermal, and performance aspects. Of course, the server models also include the individual component models stated in the previous sections. They all have in common, that the models are dependent on a single technology and refer to known working conditions, such as an application or particular operations. The models guarantee a suitable accuracy via using various instruction counter types to determine the resources' activity level or performance. The counters are hardware-specific because of the resource architecture, inner designs, states, connectivity, or parameters. Therefore, the hardware counters are only valid for an explicit configuration or platform on the micro-architectural level. For that reason, the models focus on concrete configurations within a defined environment, such as a given workload or thermal limits. The models have a range of complexity and accuracy because of the considered purpose. The approaches do not take the relationships, the dependencies, or the structures between hardware families, generations, or series into account. The models do not support forecasts on the physical and technological basis for future systems. The portability and comparability are not achieved. On the other hand, the black-box models try to abstract the hardware details to guarantee portability. Furthermore, the variable settings of workloads and software settings are crucial factors of a system's power dissipation.

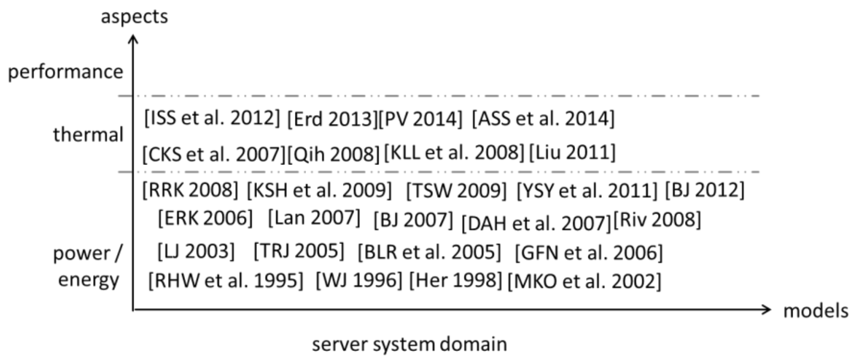

aspects
performance
thermal
[ISS et al. 2012] [Erd 2013][PV 2014] [ASS et al. 2014]
[CKS et al. 2007][Qih 2008] [KLL et al. 2008] [Liu 2011]
[RRK 2008] [KSH et al. 2009] [TSW 2009] [YSY et al. 2011] [BJ 2012]
[ERK 2006] [Lan 2007] [BJ 2007] [DAH et al. 2007][Riv 2008]
power / energy
[LJ 2003] [TRJ 2005] [BLR et al. 2005] [GFN et al. 2006]
[RHW et al. 1995] [WJ 1996] [Her 1998] [MKO et al. 2002]
models
server system domain

**Figure 39: Server system power/energy, thermal, and performance models**

We differentiate between power/energy, thermal, and performance models and their related component-based simulators. Parts of them build the foundation for full-system simulation to predict energy efficiency, for instance, and enable the opportunity for studying diverse management techniques (*how and with what to optimize*) or optimization strategies (*with respect to the optimization goal*). Furthermore, we distinguish the full-system simulations into hardware measure-based approaches, software profiling tools, and simulation frameworks. Section 3.4.2 widely describes counter-based models, such as *Mantis* [ERK 2006, Riv 2008, RRK 2008] or the *Joule Watcher* approach [Bel 2000]. These approaches give a more exact estimation about the system's behavior because of realistic workloads, including operating system effects on behavior, and performance in the working progress. The resource utilization is obtainable via measurements of synthetic benchmarks or during the runtime of the respective system. Software profiling tools simulate applications and effects, such as the operating system behavior more quantitatively. The approaches model the instruction execution after studying an application. The authors observed the timings and functional data of realistic workloads. Therefore, we call them static full-system simulators or application-level simulators. Bellosa combined hardware measurements with the software power profiling. Here, we present the software profiling tools: *SimOS*, *SoftWatt*, *Simics*, and the timing model *TFSim*. On the contrary, the *SoftWatt* approach not only has the purpose of profiling, but it also simulates the system. This is an advantage of the dynamic full-system simulators, which execute instructions with timing behavior. Those executions-driven simulators need a more exact timing model. Herein, the simulation detail and accuracy of functionality and performance are important facts to choose the adequate simulation model. We introduce the dynamic full-system simulators *SimFlex*, *SimWattch*, and *BladeSim*. A system consists of various components and physical chips and therefore, some low-level or component-based simulators are required, such as *SimplePower*, *SimpleScalar*, or *Wattch*. The academic research focuses more on the software profiling and hardware measurements at the system (component) level in comparison to the full-system simulations. In last recent years, more and more researchers concentrate on simulation-based approaches at the environment level, as shown in Figure 40. A couple of network simulations for the data center environment exist

[AP 2014, JPL et al. 2014], but they do not simulate the server system. The environmental simulation models, such as *DCSim* [SSG et al. 2009, CYR 2012, KTL et al. 2013, TKS et al. 2013] concentrate on thermal simulation. At a high abstraction level, job scheduling or management strategies consider costs, airflow, power, or energy efficiency [RAM et al. 2009, ADK et al. 2012, MJW 2012, FBP et al. 2014, KRS et al. 2014, MMA 2014], which are not the scopes of this thesis.

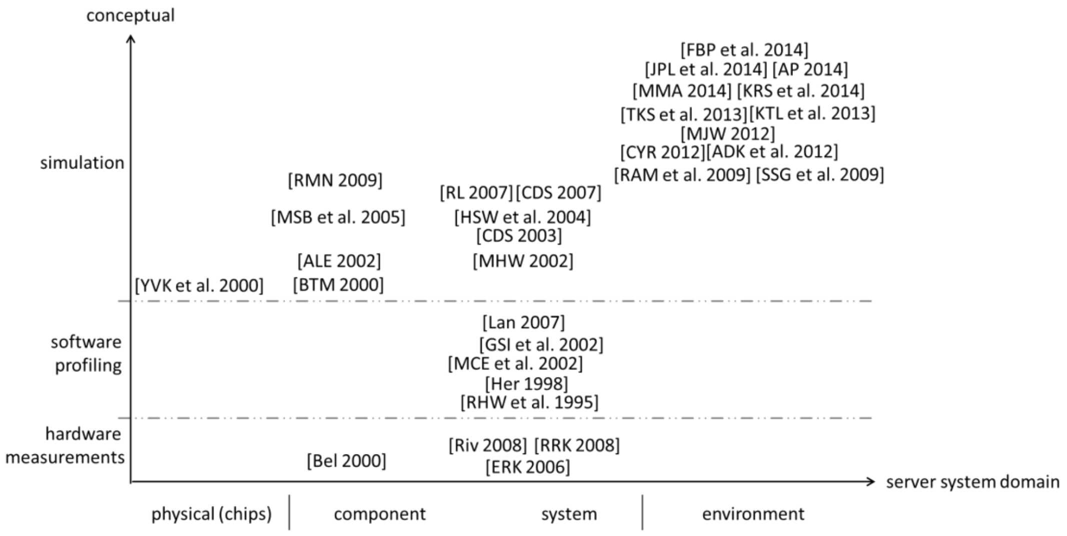


**Figure 40: Server system simulation models**

### 3.9.1 Full-System Simulation

A full-system simulation framework developed at Stanford is called *SimOS*, which is an application-based simulator that covers a variety of systems [RHW et al. 1995]. The *SimOS* system executes multiple commercial operating systems, such as *Irix*, *Tru64*, *Windows NT*, *GNU/Linux*, and their activities. The dynamic simulator supports multiprocessors, such as for *MIPS*[150] and *Alpha*[151] instruction sets supported by the processors, as well as memory models *Embra*, *Mipsy*, and the related interpreter *MXS* [Her 1998]. It includes statistics as well as working information and offers an interface that allows the customer to characterize the behavior. The authors determine the system behavior and collect corresponding data, which is a possible input for the system. The authors of [Lan 2007] extend the *SimOS* system to support better scalability and performance. They use binary translation and parallelism to model the execution of the operating system and the application software of medium-scale, shared-memory multiprocessors. The approach is restricted to the specific applications designed for an explicit hardware and the operating system. The trace-based design supports various architectures, designs, and configurations, but is based upon low-level data. Furthermore, the system can execute, complete, unmodified binary workloads as well as a variety of software.

[150] MIPS: microprocessor without interlocked pipeline stages, http://imgtec.com/mips/
[151] Alpha: http://www.alphaprocessors.com/

The *SoftWatt* approach, proposed in [GSI et al. 2002], is a power and performance simulator for the behavior of applications and the operating system. The authors quantify the power behavior with respect to the workload. They extend the *SimOS* approach to include analytical power models for various hardware components, such as the memory hierarchy and disk. The method analyzes the impact of the operating system and characterizes it at the various phases[152]. They divided the application-specific behavior, executed on a commercial operating system, into power dominant services, such as kernel activities, data path, or caches. This approach is unsuitable for a server model within the design phase because the model must be independent of the operating system. The customer has the free choice of the desired operating system[153] and this is done according to the needs of the customer's company. Some of the indicators are the company's size, such as large-scale enterprise, which influences the general budget situation, the service, and the related server type. Additionally, the model relies on the local events of the system. "Soft-Watt models a simple conditional clocking model. It assumes that full power is consumed if any of the ports of a unit is accessed; otherwise no power is consumed, [GSI et al. 2002]." The authors do not consider leakage power during idle times and in reality, the components do not consume the whole power because of power management strategies. The authors of [GSI et al. 2002] show a state machine for a full-system simulation and corresponding power values in a range between 0.15 and 4.2 watts for a hard disk drive. The power part is very simple in comparison to the other components and therefore, a low accuracy model is enough. Of course, the proportion increases by the number of drives, especially in web or file servers. The authors found that the memory's power is more than twice the data path's power. The idle power has a share of five percent in their system configuration.

*Simics* is another dynamic full-system simulator [MCE et al. 2002], which includes functional descriptions, such as instruction execution as well as timing aspects of unmodified operating systems, kernels, and drivers. The system supports the operating systems: *Solaris*, *Linux*, *Tru64*, *Windows XP,* and the related activities. The processor models are instruction set simulators for *Alpha*, *PowerPC*, *SPARC*, *MIPS*, *ARM*, and *x86-64* architectures. The system supports next-generation devices by including adjusted functional and timing behavior. The system models memory accesses and instructions with uniform time slots. Cache and memory timings are a possible extension of the system. The authors of [MHW 2002] compensate this drawback in *TFsim*[154], a full-system multiprocessor performance simulator. The approach includes micro-architectural details, such as pipeline, and models the execution of dynamic instructions within

[152] Analyzation and characterization: post-processing of the information logs from *SimOS* performance simulator

[153] Operating systems: Microsoft Windows Server (Hyper-V, Server 2x), Red Hat Enterprise Linux (RHEL), SUSE Linux Enterprise Server (SLES), Ubuntu Long-Term Support (LTS), Oracle Linux, FreeBSD, Solaris, Linux container virtualization (LXC), Xen-based virtualization, or Kernel-based virtual machine (KVM)

[154] TFSim: later known as Opal

the timing simulator. The model distinguishes between functional and timing behavior, which is decoupled to each other to consider various fidelity requirements. The timing simulator (execution of instructions) influences the functional simulator (access of register or memory) but does not change internals, such as the register. The software profiling simulators are adequate for a given environment, but do not claim to be generally applicable. Hence, the dynamic full-system simulators avoid this disadvantage.

The *SimFlex* simulation framework developed in [HSW et al. 2004] use statistical samples on a platform within component-based design. The approach includes the *Simics* features to provide operating system behavior and functional timing models. Furthermore, the authors used the *SMARTS* methodology [WWF et al. 2003] to have realistic workload samplings. The model consists of components in hierarchical manner, which are linked together with the wiring description. The definition of the ports and the control flow in C++ connects the components at compile time. The authors in [WWF et al. 2006] simplify the *SimFlex* model to speed up the simulation time. Finally, they used parallelism and checkpoint-based sampling.

On the contrary, *SimWattch* is a performance and power estimation tool that provides a complete system simulation environment [CDS 2003]. The approach combines the cycle-level, micro-architectural timing of *Simics,* the power as well as performance estimation of *Wattch* [BTM 2000], and the customer-level simulator of *SimpleScalar* [ALE 2002]. The authors focus on operating system effects, such as miss rates, to avoid misleading results at customer-level simulators. Thus, they generate instruction traces at the cycle level, which are buffered, translated, and accessed in various modules. The power model is based upon the load capacitance, supply voltage, and frequency as described in (3.22). The performance characteristics use the IPCs stated in Section 3.7. Furthermore, they analyze the OS activities, the dynamic instruction mix of workloads, the customer instructions on each application, and related cache misses per instruction [CDS 2007].

The *SimFlex* and *SimWattch* approaches require an existing system to define the component interconnections, the OS activities, and executed instructions at the micro-architectural level. The simulation tools do not scale with regard to the hardware configuration and therefore, consume much time in comparison to the real system.

Another high-level, full-system model is the *BladeSim* approach, which is proposed in [RL 2007]. It is a resource utilization-based simulator that considers the system configuration and architectural policies. The simulator converts the trace of a real workload and the corresponding resource utilization levels, which is extended by the timing model. *BladeSim* simulates the system behavior by using the correlation between the task-based resource utilization for power or performance metrics, concentrating on the processor, memory, disk, and network. The authors argued that the observation and resource utilization of the systems is straightforward because of the established trace opportunities (monitoring, control interfaces) or the simplicity to receive the values via scripts. One challenge is the variation of resource utilization in various system configurations. In addition, the key challenge is the relationship between the performance and power metrics. They found that each server class requires a specific model because of the various hardware configurations, such as the voltage and frequency states. The authors reduce the workload complexity and simulation time with their approach. "The system workloads are modeled as resource utilization traces and systems are designed as lookup tables based on the characterization functions, [RL 2007]." The approach does not consider detailed component-level behavior, buffered, or queued work. Furthermore, this method requires a real system for the characterization via calibration experiments. An alternative is an analytical model of the system with regard to the resource utilization. The accuracy of the simulator works well (the authors report a five-percent error range), in comparison to the investigated effort. In addition, the model scales to a large number of homogeneous systems with heterogeneous configurations.

### 3.9.2 Physical-based and Component-based Simulation

At the physical level, the execution-driven and cycle-accurate approach called *SimplePower* is proposed in [YVK et al. 2000]. It includes a transition-sensitive energy model that works on cycle-based executions and switching capacitance, defined in Equation (3.20). The approach consists of five stages of a pipelined data path and uses the instruction set architecture of the *SimpleScalar* method. The authors use lookup tables to store the capacitance of circuit events such as switching activities and transitions. Empirical experiments generate the data, which depend upon functional units. These analytical models are more specific rather than generic and need to be set for a specific circuit. In addition, the modeling process requires much time because of the architectural details and is a typical off-line model.

*SimpleScalar* provides an infrastructure for system and component modeling at the architectural level. The system provides an instruction set simulator for *Alpha*, *PowerPC*, *x86*, and *ARM* to execute workloads considering timing aspects. The framework provides routines to model tasks, e.g., discrete events. The execution-driven simulation uses instruction-set and I/O emulators to support customer applications, which are based upon hardware activities. The emulator maps the application to the target architecture and the functional core. The authors in [ALE 2002] describe the cache and the related timing model. The models execute the instructions at the cycle-accurate level. The approach uses baseline models that are independent of data collections, such as statistical analysis or event handlers. The very simple model abstracts micro-architectural details and uses elements such as branch predictors or instruction queues. Beneficially, it has a configurable micro-architecture to enable various evaluations.

A power analysis and modeling framework at the architectural level is proposed in [BTM 2000]. The *Wattch* approach uses the cycle-level performance simulator *SimpleScalar* and configurable power models that estimate the power and performance ratio. Herein, the processor structure is abstracted into array structures (cache, register files), fully associative content-addressable memories (logic, buffer), combinational logic (functional units), and clock-related categories (capacitance load, clock buffers). The activities are traced on a cycle-accurate level of the micro-architectural structures. This statistics form the basis for the analytical models based upon capacitance. The off-line framework allows modifying the micro-architecture by means of a table-based approach, but it is a time-consuming simulation system.

A detailed processor and memory simulation is done in [MSB et al. 2005]. The general execution-driven multiprocessor simulator (*GEMS*) characterizes the multiprocessor performance. The included timing simulator originates from *Simics* and is decoupled to the functional aspects, but the timing simulator can influence the functional behavior to capture time-dependent effects. The system consists of a random tester module, a micro-benchmark module, *Simics*, and *TFSim*. The *GEMS* toolset models various memory hierarchies, such as caches, cache controllers, or main memory. The cache coherency model uses a per-memory-block state machine, which includes states, events, transitions, and actions. The processor model provides the instructions or execution units of the *MIPS* architecture. The *GEMS* toolset does not support trace caches, hardware multithreading, or diverse ISAs.

The authors of [RMN 2009] focus on synchronization and timing for full-system network simulation of multi-computer systems and deal with the trade-off between simulation speeds, time, and accuracy. Therefore, to avoid effort they use *Simics* and its modules, which simulates unmodified operating systems. In addition, the authors combine *Simics* and the *Interconnection Network Simulation and Evaluation Environment* (*INSEE*) [RM 2005] and cover the challenges to design a full-system simulation. A main aspect is the time synchronization caused by the cycle-events of a processor, and the physical time of the network stack. The authors argued that the reuse of modules has the risk of inaccuracy and invalid results, because the modules are designed for a different purpose.

### 3.9.3 Conclusion

At the server system domain, the rigorous simulation frameworks are a mix of simulation and direct hardware measurements, such as software profiling tools, which consider instruction sets, capacitance, activities, states, transitions, events, or performance counters to estimate the behavior. The researchers limit the applicability[155] because of the published methods to work exclusively with:

[155] Limited applicability: specific constraints, preconditions, and assumptions

- A predefined simulation purpose and an explicit use case
- A fixed environment
- A target hardware platform
    - An exact system configuration
    - A restricted number of components
    - An explicit hardware specification
- A target software configuration
    - A given application, software traces
    - A defined operating system

The trace-based methods work on the cycle-accurate level, which is sufficient when another application executes the same instructions on an identical system or component. The instruction or cycle-accurate approaches [RHW et al. 1995, Her 1998, MCE et al. 2002, GSI et al. 2002, Lan 2007] would require a workload resolution higher than our 1-second to characterize the instructions and component power in a precise manner, such as by an accuracy of less than five percent. Thus, the approaches consider realistic memory read/write accesses or given tasks, which needs to be known, but the well-known instructions vary from the processor architecture, generation, or family. However, the approaches require a real system and the instruction-based trace, which is time and cost intensive, especially for unfamiliar components. Consequently, the measurement and modeling efforts increase, and these analytical models face the trade-off between the accuracy and the simulation time. In contrast, the full-system approaches on the basis of the hardware measurements study a specific hardware configuration concerning the flexible processor utilization levels. The approaches of [ERK 2006, HCG et al. 2006, FWB 2007, Riv 2008, RRK 2008] require several training techniques, which guarantee an accuracy less than ten percent. The approaches consider the linear regression methods to calculate the power that corresponds to a certain utilization level and performance counter specific to the hardware configuration. The component characteristics, such as the channel[156], the generation[157], or the technology [158] of the memory modules, have changed over the last recent years. The approaches up to now do not cover these heterogeneous characteristics of the hardware variations. In consequence, the full-system approaches are not generally valid for various server systems or tend to become impractical because the proposed models of [MHW 2002, CDS 2003, HSW et al. 2004, CDS 2007, RL 2007] concentrate on a particular system configuration and explicit operating system in which the models do not flexibly support certain characteristics. Most of the full-system approaches are not aware of the component interactions or thermal dependencies that are relevant to the industrial field of application.

[156] Channel: Dual-channel, triple-channel, or quadruple-channel
[157] Generation: DDR, DDR2, DDR3, DDR4, xxx-200, xxx-266, or xxx-333
[158] Technology: RDIMM (registered), FBDIMM (fully-buffered), or SIMM (single inline)

## 3.10 Industrial Field of Application

The academic approaches and commercial tools are part of heterogeneous product development stages. We introduce the product development life cycle that helps to explain the commercial field of application and the specific requirements. Afterwards, we describe the internal and external use cases of industrial solutions. In addition, we show the reason why the commercial tools do not consider academic approaches and we describe four common power calculators. In conclusion, we define the gap between typical industrial tools and academic approaches.

### 3.10.1 Product Development Life Cycle

Introducing a new product follows the various stages of the life cycle beginning with requirement engineering up to the sell process. The authors of [MSD 2006] define the product development life cycle (PDLC) phases: *envision* (*define)*, *build*, *test*, *implement*, and *operate*[159]. All separate sequences rely on each other, such as the predecessor phase. The development for a non-existing product starts with the *envision* phase. Herein, the vendors critically analyze existing markets to predict product requirements as well as find strengths and weaknesses. The *envision* stage includes customer feedbacks and future forecasts to develop a product vision. A recent technology roadmap includes ideas about functionality, ability, performance, or design. For instance, *Intel*'s roadmap [Int 2013] predicts new server processor families using an alternative manufacturing process technology called either *tick* or introducing another micro-architecture called *tock* [Int 2006]. New micro-architectures, changed instruction set architectures, or designs provide additionally functionality and improve the energy efficiency of the processors. The academic research and product development are part of the next phase, called *build*. The developer or manager defines the architectural requirement specifications at the technical level. Afterwards, the vendor starts the proof of concept in the early *build* phase to check constraints and feasibility. In this stage, the vendor has a first product prototype, which is also part in the *testing* phase. The developers test the software and hardware in comparison with the functional specification of each single component as well as the complete system. In this phase, academic research focuses on alternative designs and management techniques. Final acceptance tests check if the first prototypes fulfill the specifications. The product manager decides about a specification change or revision because of competitors, which leads to start partly from the *envision* phase. In the *implementation* stage, the vendor provides the complete system functionality, which is part of the system specification. Particular product lines have concrete system requirements because of a specified purpose. The *operate* phase is not precisely separated from the *implementation* phase. A customer orders an individual server system with a customized hardware and various software components. The vendors manufacture a complete system with unique customer

[159] PDLC phases: alternative identifiers are *conception*, *specification*, *new variants*, *enhancement*, and *maintenance*

requirements and specifications. The order process and proper product documentation are the main part of the *operate* stage. If a product is at end of life (EOL) or end of sale (EOS), the life cycle will start again with a new or adapted product. Figure 41 shows the complete product life cycle.

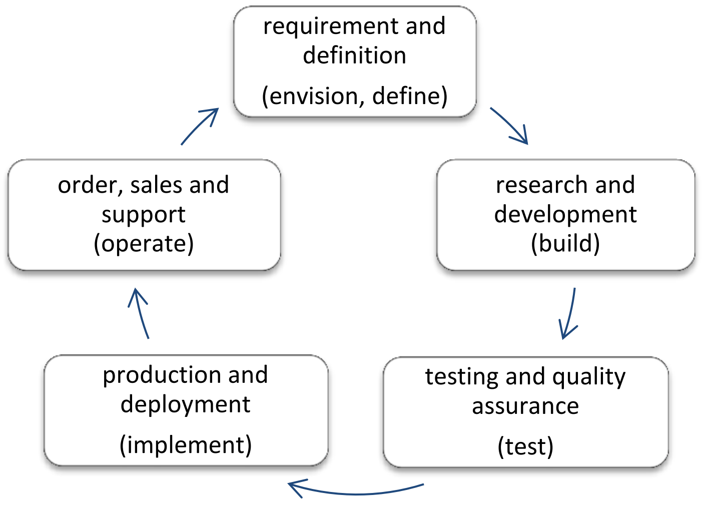


Figure 41: Product development life cycle [MSD 2006]

The academic approaches and commercial tools partially support at least one of the five product development life cycle stages. The following section describes the field of application of the academic and industrial approaches.

### 3.10.2 Research, Development, and Deployment of Server Systems

On the one hand, the academic approaches offer an unknown perspective to the established practice and provide optimized algorithms to integrate new functions as well as features that are part of the product life cycle *build* stage. On the other hand, the commercial tools do not include the systematic findings and results. The established industrial tools offer the opportunity to purchase a server system, which is the common business case in the *operate* stage. The customer configures a server suitable to match the customer requirements.

The requirements differ between superficial and deep knowledge, depending on the server system domain described in Section 3.3. A data center manager plans the overall energy demand and in comparison, a technician is aware of the temporary power constraints within a rack enclosure. Another power-related decision is the server type, because a high-end server consumes more power than a mainframe, or an ordinary server. In addition, the enterprise sizes predefine the server enclosure because of the provided space and capacity. Data centers contain various server kinds, such as storage servers or high-performance servers, which have special configuration demands. The unlimited variability of the customer induces the vendor to establish flexible and universal (but still vendor-specific) configuration tools that satisfy the diverse demands. Thereby, the commercial tools focus on the system specification phase to release a product. At the same time, they do not apply academic approaches, because of the high measurement and modeling effort investigated for each single component. Academic researchers focus on an explicit scenario to keep products competitive, but they use a predefined configuration. An important consideration in ensuring the order process is a flexible hardware configuration. The executed software on the server varies for each customer and is a subordinated interest.

Furthermore, the vendors use commercial tools to win the invitation to tender. In this case, a customer provides a clear guideline on the hardware required in their enterprise and creates a call for tenders with many individual positions. For instance, the customer may require an *Intel Xeon* processor (*E55xx*), $24GB\ DDR3-1066$ memory, 2-terabyte hard-drive capacity, and a Linux Server OS (Red Hat Enterprise Linux – RHEL or SUSE Linux Enterprise Server – SLES) by the 500 watt power limit and the upper limit on costs at 700€ per server. The customer does not specify the intended purpose and therefore, the vendor calculates the peak power consumption on the basis of the technical specification. Instruction-based or cycle-accurate approaches do not work, because of the missing data about the range of tasks or precisely defined jobs. Furthermore, commercial tools must be independent of the operating system, whereby most academic full-system simulators refer to a given one [GSI et al. 2002, MCE et al. 2002, HSW et al. 2004]. Here, we consider that the server does not pass the power limit under any circumstances, because of the regression claims [Fuj 2011]. The server vendors assure commitments, such as technical specifications, that form the legal framework. In industrial practice, theoretical data, worst-case thresholds, and over-estimation ensure the product specification besides management techniques. In general, the customer does not explicitly mention something about environment conditions, such as the ambient temperature, thermal, or performance limits of the server system. Data centers work with a thermal range from $20°C$ up to $35°C$ for which the vendors specify the system. If there is a given environment limit, the vendor only checks if the system is within the specification.

Vendors deploy a server to a customer into the production environment and offer maintenance services. In this phase, the server behavior may include bugs or problems, which the customer reports to the vendor. In general, such reports are about the reliability, availability, maintainability, and safety (RAMS). Vendors solve the reported issues via a customer-specific bug fix, a server system adoption, or academic research for the next server generation. The server operates many times larger than the development time. The software solutions are cheaper options in comparison to hardware changes. Therefore, vendors investigate firmware-based solutions to support a range of servers at the same time. If firmware changes do not satisfy the demand, a vendor has to adapt the hardware. Academic research indicates the related software as well as hardware changes under predefined conditions provided by the customer. The particular use cases of the server system provide the environmental conditions, such as the operating system and the workload, which results especially in the research question. A rework of the entire system saves the investigation costs and ensures more profit for industry in comparison to a complete new server system. The individual server systems share the high development costs. In addition, a vendor includes novel market requirements and forecasts.

Various use cases result in multiple perspectives of an industrial tool. Table 18 compares the customer and vendor demand, which both in common require a flexible hardware and software configuration for the server. On the one side, the customer chooses an explicit hardware and software configuration to fulfill his or her own performance demands. On the other side, the vendor supports already released systems in a given configuration subset. Our simulation tool should offer the ability for future systems and server variants that are in development. The flexible software settings require a realistic workload scenario. At the same time, full utilization of all resources constitutes the worst-case scenario. The executed jobs are unknown in the vendor perspective and therefore the tools abstract the software by means of resource utilization levels. Additionally, vendors do not consider the operating system as well as virtualization techniques for customer-specific jobs. The requirements of customers and vendors do not match, because customers require an operating system that considers management techniques, virtualization, or optimization strategies. The customer specifies the same job that is already applied in the obsolete system. The system requirements come from the real application and the customer's daily experiences. On the contrary, the customer-specific job may be unknown because the operations and tasks change within the future. Another non-predictable aspect is the job scheduling by the data center manager, which utilizes the resources in any manner. The customers use the commercial tools to calculate the power of the entire server system. On the contrary, vendors additionally consider the component power consumption and the PSU sizing. In the commercial context, a vendor calculates the server power for a worst-case scenario, including an extra power overhead [Fuj 2011]. This method ensures that the PSU has the adequate power under every circumstance, and the server has no need to throttle down, which may reduce the performance. Nevertheless, customers require an optimally utilized PSU, which is more efficient for 80% instead of 50% utilization. The over-provisioning also covers future demand, because a vendor designs a system for three up to five years. In the case of large-enterprises, data centers use a server much longer. A vendor calculates the power demand of a server system, which is fully equipped in the future. The power supply offers sufficient capacity to ensure the incremental growth of electronic devices within the server system. The energy efficiency consideration is not yet part of industrial tools, but included in all academic research. The green IT trend supports the demand of energy-efficient servers, which leads to the energy efficiency requirement within the commercial tools. Therefore, customers ask for information about application-based power (energy) consumption and the related temperature as well as performance. The commercial tools do not follow this trend to our knowledge and provide standard data about peak power, airflow rate, noise, or heat emission. Standardized benchmarks show the absolute performance values of a server system, which are not part of the industrial tools.

Table 18: Comparison between customer and vendor requirements of the industrial tools

| | Customer | Vendor |
|---|---|---|
| **Configuration** | Hardware, software, flexible | Hardware, software, flexible |
| **Hardware settings (change components)** | Explicit configuration (customer-specific) | Released systems and configurations, future systems, new variants |
| **Support non-existing server** | Not required | Internal for *build* stage |
| **Software settings** | Predefined configuration | Flexible |
| **Workload scenarios** | Customized, actual workload | Realistic scenarios, worst-case (fully-utilized resources) |
| **Executed jobs** | Unknown, real applications, actual job, future jobs, or scheduled tasks | Unknown (black-box), abstraction from software and operating system, customer-specific |
| **Virtualized workload** | Consider virtualization | Independent |
| **Operating system** | Customer-specific, management techniques, optimization strategies | Independent, neglect |
| **Power calculation** | Server systems | Server systems and components |
| **Power supply unit** | Suitable, efficient, optimistic assumption | Pessimistic assumption (worst-case), extra buffer |
| **Over-estimation** | Reduction of over-provisioning, minimize worst-case assumption, save operational costs | A range of added power that is not used |
| **Energy efficiency** | Energy efficient (optimal) configuration | Customer-specific, flexible |
| **Power / energy** | Customer-specific, application-based | Peak and actual power |
| **Thermal** | Temperature behavior, peak value, airflow rate | Airflow rate, noise, heat emission |
| **Performance** | Customer-specific, absolute data | Customer-specific, relative data, via benchmarks |

The following section describes the industrial power calculation tools, which do not use academic fundamentals because of the various requirements and use cases.

### 3.10.3 Commercial Server System Calculators

In industrial practice, server vendors, such as Dell[160], IBM[161], HP[162], or Fujitsu[163] provide web-based or application-based power calculators for server systems. On the one side, the customer configures the entire system for his or her needs and calculates the server power consumption to be aware of the overall energy demands of the IT-infrastructure. On the other side, the configuration tools offer the opportunity of generating the bill of the material lists (BOM) to order a system. The configuration tools propose large quantities of up-to-date as well as earlier (archived) server systems, enclosures, related infrastructures, and their components. Table 19 summarizes the settings of the industrial tools that calculate the power consumption of the complete server system and shows the vendor-specific thermal outputs. The table cell content indicates if the input settings are supported (via yes) by the tools of the *Dell Energy Smart Solution Advisor* (ESSA)[164], the *Fujitsu System Architect*[165], the *HP Power Advisor*[166], or the *IBM Energy Estimator*[167]. They all have in common that the hardware configuration, such as the processor, memory, and I/O is highly flexible by choosing the type, family, quantity, or capacity. The IBM tool only distinguishes in the number of active processor cores, which leads to the assumption that the approach of the equation (3.31) is used. The tool calculates the power consumption of the entire system, whose values are non-linear with respect to the configured active cores. Dell exclusively distinguishes the memory-working mode as an input parameter, but the system power dissipation does not change at various settings. All vendors besides IBM consider power supply redundancy. Either one or more PSUs provide the system's power to avoid AC losses or power failures. In addition, it could be more efficient to use multiple PSUs at $50\%$ utilization level instead of one PSU at $100\%$ utilization level. The 80 PLUS® performance specification[168] certifies PSUs whose energy efficiency is over $80\%$ and higher (symbolized with bronze, silver, gold, platinum, or titanium) at all utilization levels. For instance, a titanium-certified PSU has an efficiency of $90\%$ in all utilization levels and therefore, the power supply redundancy has no effect on the power consumption when only the amount of the same PSUs changes. With the present state of our knowledge and general experience, the bronze-certified power supply units are state-of-the-art. The fan settings are vendor-specific and are impossible to be changed for a customer-specific thermal control. The customer cannot directly influence the server's noise, airflow, or temperature behavior. Furthermore, the tools provide the choice of the software, such as an explicit

[160] Dell: Dell Inc., http://www.dell.com/
[161] IBM: IBM Corporation, http://www.ibm.com/en-us/
[162] HP: Hewlett-Packard Development Company, L.P., http://www8.hp.com/us/en/home.html
[163] Fujitsu: Fujitsu Limited, http://www.fujitsu.com/global/
[164] Dell: http://essa.us.dell.com/DellStarOnline/DCCP.aspx?c=us&l=en&s=corp&Template=6945c07e-3be7-47aa-b318-18f9052df893
[165] Fujitsu: http://configurator.ts.fujitsu.com/public/
[166] HP: http://www8.hp.com/us/en/products/servers/rackandpower.html#poweradvisor
[167] IBM: http://www-912.ibm.com/see/EnergyEstimator
[168] 80 PLUS PSU: http://www.plugloadsolutions.com/80PlusPowerSupplies.aspx

application, virtualization, or operating system. This is only part of the material list and not part of the power calculation. The same effect occurs for the BIOS/UEFI settings in the Dell tool[169]. All four tools support the power calculation on the basis of the processor utilization level. The Dell calculator additionally provides a memory and computational configuration, but we observed that the power calculation is fixed. In the case of the computational and memory-bounded workload, the related power comes from the power consumption at the highest processor utilization $u_p = 100\%$. Equations (3.81) and (3.82) show the weight factors in relation to the processor-bounded workload. The factors come from empirical experiments that use various server configurations (30) to analyze the influence of the settings.

$$p_{computational} = 1.07 * \, p_{processor\_bounded}\big(\max(u_p)\big) \tag{3.81}$$

$$p_{memory\_bounded} = 0.94 * \, p_{processor\_bounded}\big(\max(u_p)\big) \tag{3.82}$$

Furthermore, Dell uses the static power consumption for the idle utilization level as stated in the equations (3.19). The calculator uses the thermal design power of the processors to estimate the largest power consumption. The server system power is linearly proportional to the processor utilization, and the number of processors used within the system. The linear regression is valid for the other vendor tools as well, which we checked in a server system evaluation of the *Dell R720*, *Fujitsu RX300S7*, and *HP DL380p Gen8*[170]. The use case of the calculators is the same, so we select almost the same hardware configuration (technical specification) to achieve comparable results. In our evaluation, the systems use the dual socket processor of the Romley EP platform, the equal amount of memory, and a nearly identical storage capacity. We observed that all tools use a linear algorithm to calculate the server power consumption. A higher amount of processors results in a taller power-to-utilization curve gradient, but is still a linear relationship. Furthermore, we found that the offset to the base power at 0% utilization level only relies on the processor quantities. In addition, the IBM tool handles the power save mode in the configuration, but this does not have any impact on the power calculation. The power tools of Dell and Fujitsu calculate the airflow rate of the server system, which based upon the enclosure conditions or limitations. The Fujitsu tool estimates the heat emission in a simplified manner, which relies on the statement that every consumed power generates the same amount of heat. On the contrary, the Dell calculator provides the noise instead of the heat, which is another optimization aspect.

[169] BIOS/UEFI settings: only Dell supports customer-specific settings
[170] Evaluation: *IBM X3650 M4* was impossible to check, because of access rights

Table 19: Industrial server systems power calculators (Dell, Fujitsu, HP, and IBM)

| | Dell | Fujitsu | HP | IBM |
|---|---|---|---|---|
| **Input:** | | | | |
| Hardware variation | | | | |
| Processor | yes | yes | yes | yes |
| Number of active processor cores | - | - | - | yes |
| Memory | yes | yes | yes | yes |
| Memory operation condition | yes | - | - | - |
| I/O (hard disk drives, network) | yes | yes | yes | yes |
| Power supply | yes | yes | yes | yes |
| Power supply redundancy | yes | yes | yes | - |
| Fan | - | - | - | - |
| Software configuration | | | | |
| Application | yes | yes | - | - |
| Virtualization | yes | yes | - | - |
| Operating system | yes | yes | - | - |
| BIOS/UEFI settings | yes | - | - | - |
| Workload | | | | |
| Processor-bounded (CPU utilization) | yes | yes | yes | yes |
| Memory-bounded (intensive) | yes | - | - | - |
| Computational | yes | - | - | - |
| Environment condition | | | | |
| Ambient temperature | yes | yes | - | - |
| **Output:** | | | | |
| Power consumption at CPU utilization level [0 - 100%] | yes | yes | yes | yes |
| Enabled power save mode | - | - | - | yes |
| Airflow rate | yes | yes | - | - |
| Heat emission | - | yes | - | - |
| Noise | yes | - | - | - |

### 3.10.4 The Gap between Typical Industrial Solutions and Academic Approaches

The industrial server calculation tools offer the same key features, such as configuring the server at the hardware and software level to calculate the server power consumption at an exact processor utilization level and, finally, order a system. The power consumption is a static value due to the demand at a specific time. No tools support customer-specific scenarios or applications that refer to a realistic behavior over time. It is impossible to configure the server workload in a flexible manner. With the present state of our knowledge and related research papers, the tools use measurement-based and spreadsheet-based databases that refer to a specific software and hardware configuration [Fuj 2011].

Table 20, Table 21, and Table 22 represent the differences between industrial practice and academic research for full-system server power calculators and simulators. In comparison to the academic approaches, the industrial power tools support a range of server systems and configurations, such as the amount of processors and choosing the vendor, the family, or the generation. The tools work with diverse mandatory and optional configurations. On the contrary, the approaches in Section 3.9 focus on an exact system or component architecture, which supports the tested instructions for the functional or timing model. The academic models do not abstract from the operating system or application and therefore, the authors use an explicit hardware setting for their method. Both approaches do not consider the device operation condition, which in specific terms is a hardware-based setting only configurable when the system is in the BIOS/UEFI state. In the BIOS/UEFI, for instance, the customer enables the virtualization-based mode of the processor or the sparring or mirroring mode of the memory modules. During runtime, the behavior is fixed and must be set before the system starts. In addition, the management techniques cannot influence these base decisions, but the BIOS/UEFI settings are not negligible for the power consumption of the system. In industrial tools, the customer selects an operating system or application, which the power algorithm does not consider. On the other hand, the researchers analyze the systems on the basis of the OS types, working sequences, tasks, or threads because of their influence on the power consumption. Therefore, the authors cover a range of benchmarks, applications, or scenarios executed over time. On the contrary, all industrial power tools use a processor-bounded workload for a simplified power estimation of the server system at a time. The tools do not support flexible workloads to estimate the minimal, average, and peak power/energy. Neither the industry nor academic power algorithms consider the environmental conditions, such as the ambient temperature. In the case of the academic algorithm, the customers choose a management technique or strategy, which is the research objective. On the contrary, the industrial tools ignore them because the operating system (e.g. Linux or Windows) handles the performance demand and, finally, the frequencies autonomously when no management rules are given. Furthermore, the customer can set the optimization goal within the baseboard management controller, which is independent of the OS. For instance, the customer configures a power limit of 200 watt for the server system. The embedded controller limits the highest p-state of the processor to fulfill the power demand or reduce the fan speed. The software settings of the governor or OS are also able to throttle the system if it is part of the settings. On the other hand, the data center manager[171] may control the power limit and the amount of job decrease by a policy to reduce the power consumption. A server vendor has no knowledge of the environmental constraints and therefore, does not consider management techniques. The main aspect of industrial tools is the power calculation, but at the same time, the algorithms do not include the thermal related power that occurs from the fan speed control.

[171] Data center manager: differentiate in business unit manager, IT application manager, IT hardware manager, or data center facilities manager [PBB et al. 2010]

We found that the industrial power algorithms only rely on the processor as well as resource utilization and not, as in the academic case, in instructions, clock cycles, bandwidth, accesses/misses, states, transitions, activities, or counters. Therefore, commercial tools use measurements and spreadsheet models. The academic research approaches are more complex, because the inner behavior is part of the model. The state machines are sensitive to the functional model (event, activities, accesses, or misses) as shown by empirical experiments. The academic algorithms offer many calculation behaviors beginning with an additive relationship up to exponential algorithm. In contrast, all industrial tools calculate the power using a linear regression for a time. The average error between the estimation and measurement of the industrial tools is more than double $(> 20\%)$ in comparison to the academic approaches $(< 10\%)$. The impact of the academic results is a higher measurement as well as modeling effort that consumes more time and costs. Therefore, the vendors set their own optimum between the level of detail, accuracy, measurement, and modeling effort. Fujitsu stated an error rate of approximately ten percent for full-system power consumption is adequate in comparison to the investigated effort, and the customers' requirements [Fuj 2013]. Of course, the customer itself wants the most precise results. In the phase of shorter server product lifecycles of design and development, very often no physical hardware is available. Designing a server in industrial practice considers the supplier data sheet. The component-based values are theoretical data or worst-case thresholds in order to not fear regression claims. Server vendors believe the technical specification and neither verify nor validate the values because of the time or costs that are necessary to evaluate the deliveries. Vendors calculate the total system power imprecisely because of the deviation between measured and data sheet values. This trend of error propagation continues with the number of each component within a server system, which leads to the oversized server PSUs. A correct operating state of the power supply is impossible because the PSU almost never reach a utilization level over 80 percent. Often, the vendors optimize the power supplies for peak conversion at high utilization, which leads to inefficiencies at the lower level.

**Table 20: Comparison of the power calculators of server systems in industrial practice and academic research $(I)$**

| | **Industrial practice (calculators)** | **Academic research (calculators, simulators)** |
|---|---|---|
| **Input:** | | |
| **Hardware configuration (flexibility, expressiveness)** | Variable, parameterized, customized, generic (quantity, vendor, family, capacity, etc.) | Explicit and static (predefined) |
| **Device operation condition / mode** | Yes* (energy saving mode) | - |

*: Possible to change, but not considered in the algorithm

**Table 21: Comparison of the power calculators of server systems in industrial practice and academic research ($II$)**

| | **Industrial practice (calculators)** | **Academic research (calculators, simulators)** |
|---|---|---|
| **Software variation** | Yes* (OS, application) | OS, parallelism, tasks, threads, virtualization, schedule, compiler settings |
| **Workload** | Processor-bounded**, specific, point in time | Various benchmarks, applications, scenarios, generic, point in time, over time |
| **Environment condition** | Ambient temperature* | - |
| **Management technique / optimization strategy** | -*** | Dynamic power management (DPM), advanced power management (APM), dynamic voltage frequency scaling (DVFS) e.g. *AMDs PowerNow* or *Intel Speed Step*, power budgeting, capping, limiting, throttling, gating, shifting, operating mode control, system power management, dynamic thermal management (DTM), fan speed control, resource management, allocation, planning, load balancing, workload management, consolidation, migration |
| **Output:** | | |
| **Aspect / use case** | Calculation of power** | Calculation or simulation of power, thermal, performance, or energy efficiency |
| **Scenarios** | Actual and peak values | Actual, min, peak, average, sum, optimization, dynamic behavior |
| **Focused product life cycle phase** | External: build, operate internal: envision, build, test, implement, operate | Test |

*: Possible to change, but not considered in the algorithm

**: Based on our knowledge, it is the primary property, but there are few exceptions

***: Not configurable, automatically done within the system (via OS, BMC, and data center manager)

Table 22: Comparison of the power calculators of server systems in industrial practice and academic research ($III$)

| | Industrial practice (calculators) | Academic research (calculators, simulators) |
|---|---|---|
| **Power consumption algorithm** | | |
| **Basis** | CPU utilization**, OS independency | Resource (CPU) utilization, OS dependency, instructions, clock cycles, active cores, parallelism, customer-level metrics, open sockets, bandwidth, accesses, or I/O rate of: memory, hard disk drives, network |
| **Level of detail** | Resource utilization | Instruction, cycles, voltages, frequencies, states, transitions, events, activities, counters, accesses, misses, correlations |
| **Modeling style (base of equations)** | Lookup tables, spreadsheet-based, measurement-based, configuration-based, black-box | State machines sensitive to functional model, states controlled by scenarios measurement-based (counters, profiling), simulation-based, optimization-based, white-box and gray-box |
| **Calculation** | Linear | Linear, non-linear, square, cubed, exponential |
| **Power dissipation** | Peak and actual, time | Min, peak, average, actual, over time |
| **Over-estimation (accuracy, error)** | High ($> 20\%$ average) [Fuj 2013] | Low ($< 10\%$ average) [RRK2008, FWB 2007, HCG et al. 2006] |
| **Modeling effort** | Depends on the accuracy, based on components | Based on functional or timing behavior, new for every exploration |
| **Measurement effort** | High (each system) | Extreme high (each instruction or function) |
| **Time effort, cost** | High (one system, three month [Fuj 2011]) | High (analyze and divide system activities, trace system, and instructions) |

*: Possible to change, but not considered in the algorithm

**: Based on our knowledge, it is the primary property, but there are few exceptions

***: Not configurable, automatically done within the system (via OS, BMC, and data center manager)

The gap between the commercial practice and analytical research comes primarily from the heterogeneous use cases. Academic approaches have less flexibility in the system configuration in comparison to the industrial tools. On the other hand, the commercial algorithms use restricted workloads, which are not able to consider academic results because of the missing details. Lower power calculation accuracy leads to over-estimation of the server, but requires less measurement effort than the academic approaches.

## 3.11 Summary

In this section, we present the basic modeling techniques that are used in academic research and industrial practice. We distinguish the basic modeling techniques into the object, control, and process specification of the following modeling domains: physical, conceptual, contextual, and external. The characteristics of the model and its objectives, such as the measurement-based, simulation-based, configuration- and optimization-based model, are relevant factors to choose the suitable technique. We present the following server system domains: data center, rack enclosure, components, chips and their aspect-based metrics as well as benchmarks. In this thesis, we concentrate on modeling a server system on the basis of the fundamental techniques. We present the aspect-based algorithms and approaches in academic research.

In fact, power consumption is a key factor across all domains and thus, we begin with the related approaches of the server system domain. We define the logical description of the chips or components, such as their electrical behavior, which is part of the physical domain. Usually, the interior component structures, operations, processes, instructions, and particularly the internal data are known because of the prior measurements on the real system. The timing models that consider an application and its behavior are typically used in the chip and system domain for micro-architectural description. In contrast, the approaches in the component or system domain rely upon the hardware activities or micro-architectural events, such as accesses or switching activities. These approaches specify states and transitions considering counter-based heuristics, e.g. hardware performance counters. If no other inner information about the interior behavior or functionality is available, we consider data sheets that provide the power consumption values on the architectural level concerning the component characteristics, such as the bandwidth, organization of the device, cores, or supply voltages.

The utilization levels are one of the most commonly used metrics in the models across the domains. The utilization levels are the key elements for the workload definition to build up a realistic scenario and indirectly affect the aspect-based models that highly depend upon the performed task in the server system domain, whereby the performance counters represent the chips, components, or system activity. The utilization level of the physical domain focuses on instructions and operations on an architectural level, for instance. The resource utilization results not only in power consumption, but also leads to thermal dissipation.

The thermal behavior relies on the operating condition of a system, its resources, chips, and their related states. In academic research works, several thermal algorithms consider the volume flow, density, heat, heat flow, thermal conductivity, thermal gradient, heat transfer rate, and thermal resistance. Usually, academic approaches assume a linear relationship between the component utilization, and the temperature specified as an analogy between electrical and thermal properties. Moreover, we present some next-generation approaches that define the dynamic thermal behavior of a component as an exponential time-based function considering the hardware performance counters. Current approaches for thermal modeling of the memory modules are based upon self-heating, spreading heating, or adjacent heating resistances, but do not consider the thermal interaction between resources from the same type, from other types, or inner dependencies, because of the circuit structure as well as the architecture. Current processor models only consider local hotspots, whereby the temperature is non-linear to the utilization level and the input voltage, but linear to the frequency. The simulation-based approaches require real servers for the temperature measurements of each component under static and dynamic considerations. The system-wide thermal effects are not taken into account up to now.

Besides the power and thermal models, we require the performance models to specify the energy efficiency. The performance models estimate the peak value, the average values, or the performance per watt on the basis of resource characterization to indicate how efficient the system operates. The system executes a set of workloads, such as benchmarks (suite) and the results of these empirical studies form the basis for the performance models, which depend on the executed instructions (floating-point, integer) and related activated functional units on the explicit hardware. In case of current memory performance modeling approaches, the architecture, the data locality, and the cache misses are significant criteria. In addition, the workload type generates performance differences because of parallelization or shared accesses. A forecast or prediction is only possible for an explicit workload, where the read-to-write ratio of the memory modules is established. In this thesis, we focus upon the power and thermal models and therefore consider the measurement results of several performance scores. The energy efficiency (performance-to-power ratio) is a criterion to determine the productivity, whether in power/energy, thermal, or performance. The efficiency provides the ability to avoid costs, airflow, humidity, carbon footprint, efforts, energy, time, or money at the considered aspects.

Across the various levels of detail, the aspect-based models build the fundamentals of the full-system models, which we distinguish into simulation-based and hardware-metric-based models. The system-based models depend on a single technology and refer to known working conditions, such as an explicit application or particular operations. The variable settings of workloads and software settings are crucial factors of a system's power dissipation. The models focus on concrete configurations within a defined environment, such as a given workload, and guarantee a suitable accuracy by using various instruction counter types to determine the resources' activity level or performance. The counters are hardware-specific because of the resource architecture, inner designs, states, connectivity, or parameters. Therefore, the hardware counters are only valid for an explicit configuration or platform on the micro-architectural level. The dynamic full-system simulators or application-level simulators (SimFlex, SimWattch, BladeSim) give a more exact estimation about the system's behavior because of realistic workloads, including operating system effects on behavior, and performance in the working progress. The resource utilization is obtainable via measurements of synthetic benchmarks or during the runtime of the respective system. Software profiling tools (SimOS, SoftWatt, Simics, TFSim) simulate applications and effects, such as the operating system behavior, and model the instruction execution after studying the timings and functional data. The software profiling simulators are adequate for a given environment, but do not claim to be generally applicable. The execution-driven simulators and cycle-accurate approaches at the physical level (SimplePower, SimpleScalar, Wattch) need a more exact timing model because they consider the instructions with their timing behavior, such as memory accesses and instructions with uniform time slots, cycle-based executions, pipelined data path, and switching capacitance. The approaches provide an infrastructure for system modeling at the architectural level and generate instruction traces at the cycle level. The execution-driven simulators provide routines to model tasks, e.g., discrete events, and use instruction sets, and I/O emulators to support customer applications, which are based upon hardware activities.

The full-system simulation frameworks SimOS and Simics are application-based power and performance simulators that model the execution of the operating system and the application software of medium-scale, shared-memory multiprocessors. These models are restricted to the specific workload designed for an explicit hardware and include functional descriptions, such as instruction execution as well as timing aspects of unmodified operating systems, kernels, and drivers. Furthermore, the models quantify the power behavior with respect to the workload into power dominant services, such as kernel activities, data path, or caches. The trace-based design supports various architectures, designs, and configurations, but is based upon low-level data. BladeSim is a resource utilization-based simulator that considers the system configuration and architectural policies. The simulator converts the trace of a real workload and the corresponding task-based resource utilization levels, which is extended by the timing model and simulates the system behavior. Figure 42 presents the investigated server system simulators concerning their conceptual and server system domain.

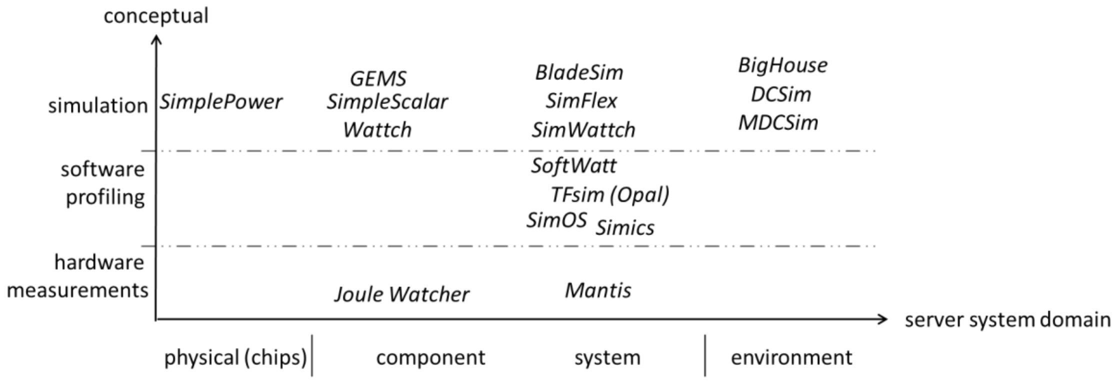


Figure 42: Server system simulation approaches

The full-system approaches require an existing system to define the component interconnections, the OS activities, and executed instructions at the micro-architectural level. The approaches do not take the relationships, the dependencies, or the structures between hardware families, generations, or series into account. The impact of the academic results is a higher measurement as well as modeling effort that consumes more time and costs. Most academic full-system simulators refer to a given operating system, and focus on an explicit scenario, and predefined configuration to keep products competitive, but do not scale with regard to the hardware configuration. In commercial tools, the full-system simulators are unsuitable for the server modeling because the model must be independent of the operating system. The customer has free choice concerning the desired operating system and server characterization according to the needs of their company.

The customer chooses an explicit hardware and software configuration to fulfill his or her own performance demands of the server system. The system requirements of the server modeling come from the real application and the customer's daily experiences. The executed software on the server varies for each customer and is of subordinated interest. Moreover, the customer-specific job may be unknown because the operations and tasks change over time. The unlimited variability of the customer induces the vendor to establish flexible and universal (but still vendor-specific) configuration tools that satisfy the diverse demands. Both in common require an adjustable hardware and software configuration for the server.

In addition to the main task of purchasing an individual server system, the vendor offers the opportunity to calculate the power (energy) consumption of the server system to match the customer requirements, such as power constraints in the server environment. In industrial practice, a vendor calculates the server power for a worst-case scenario (a fully equipped server system), including an extra power overhead in order to not fear regression claims. The component-based values are theoretical data on the basis of the technical specifications. Vendors calculate the total system power imprecisely because of the deviation between measured and data sheet values. This trend of error propagation continues with the number of each component within a server system, which leads to oversized server power supply units. We found that the industrial power algorithms only rely on the processor as well as resource utilization and not, as in the academic case, in instructions, clock cycles, bandwidth, accesses/misses, states, transitions, activities, or counters. The academic research approaches are more complex, because the internal behavior is part of the model.

# 4 Problem Statement, Challenges, and Aims

Industrial tools require the same accuracy, an error rate less than ten percent, as the academic approaches already offer. The industry is not able to use the precise concepts as they stand because of the differences concerning requirements, specific configurations, and modeling aspects. Some of the challenges are as follows:

- Vendor vs. customer perspective
- Upper vs. lower level of abstraction
- OS independence vs. explicit OS settings
- Processor-bounded workload vs. given benchmarks (software settings)
- Generic vs. particular model
- Inaccuracy vs. precision

Industrial tools do not consider instruction-based or cycle-accurate details on the physical domain. The loss of accuracy leads to an over-provisioning and inefficiency of the server system and this missing precision affects everyone who purchases a server system. A server system consumes power and generates heat at the same time. The thermal dissipation requires extra ventilation facilities to cut the heat, which is a critical issue, especially in large-enterprises, where the heating, ventilation, and air-conditioning technology create operational expenditure [WK 2013].

In the academic context, a generic approach can reduce the modeling effort, which supports different system variations without creating a new representation. The measurements describe the academic observations across a complex and entire system. Hence, another benefit is the prevention of extra measurements because of the constraints given by each customer. We need a configuration-based model that provides various hardware and software architectures as well as a behavioral description. Furthermore, the operating system abstraction offers the opportunity to use the model for particular technologies. In addition, the model should be valid under virtualized and non-virtualized circumstances, whereby each operating system is possible. The authors of [LJ 2003, DAH et al. 2007] create a simulation-based, full-system timing model at the micro-architectural level, but the functional model works on a concrete OS.

To our knowledge, in the academic literature there are no generic approaches that cover the full server system simulation on a common base. The server power depends on the configured hardware and software. The same benchmark generates nearly doubled power when we duplicate the processor quantity, but the extra resource does not double the performance. Of course, the main consumer of the system must be the processor; otherwise, this statement is not valid. The performance is less because of the component interconnection, as the authors of [SXC et al. 2000, MNR 2007, MHS et al. 2009] have analyzed. If the same hardware configuration executes different benchmarks, the peak performance or power also varies.

The researchers [Lan 2007, KSH et al. 2009, KKK et al. 2012, KJC et al. 2014] found a couple of reasons, such as the instruction type (floating-point vs. integer), the parallelism, the architecture, and the limited bandwidth.

The main concept is the integration of the academic approaches into industrial tools to close the gap between measurements and power calculation. A challenge of this thesis is the match between the application space and the architectural space of a complete server system. We have to map the workload to the server system using utilization-based scenarios. The benchmarks do not reflect the reality and therefore, there is a fundamental risk of misinterpretation of the power consumption. On the other hand, our model requires a sufficient level of detail in the configuration space, allowing the relevant energy efficiency factors of the physical domain to support the necessary precision. The authors of [SM 2001] propose the interaction between the upper and lower level abstraction, the impacts of which are the constraint propagation and performance estimation. We found various problems beginning at a higher abstraction level (environment domain) towards a minor abstraction level (physical domain). The next sections describe the findings and recommended actions.

A simple porting of the academic approaches into the commercial tool does not work. This section describes the various problems observed in academic and industrial practice. Furthermore, we address the impact of not solving the problem and the related aim. Finally, the evaluation factors support the decision to check whether the approach is sufficient. The requirements chapter describes the expanded simulation tool demands and the resulting evaluation criteria.

## 4.1 Workload Limitations

Commercial power calculators do not consider different application types, which use the resources in various ways. Industrial tools limit the workload of the server system towards the processor-bounded utilization levels. The customer is not able to create a specific workload scenario to assume different server usage and cannot receive a precise power consumption of the entire server system. In addition, the workload varies according to the customer demand and is not always a processor-bounded scenario. For instance, the *SPECpower* benchmark is a computational job that primarily utilizes the processor. Another example is *SPECjbb*, a Java server benchmark that proves the performance of a Java application. Both benchmarks offer different performance and power results because on the one side, only the processor is in use and on the other, the *SPECjbb* additionally utilizes the memory modules. Table 23 shows four benchmarks used in the Mantis approach, whereby the authors of [Riv 2008] distinguish between various components-based utilization levels. We need to differentiate between the workloads to predict precise the power consumption for a specific workload.

Table 23: Comparison of utilization levels in Mantis [ERK 2006]

| Name | Processor utilization | Memory utilization | Disk utilization |
|---|---|---|---|
| **SPECint** | Very high | High | Very low |
| **SPECfp** | Very high | High | Very low |
| **SPECjbb** | Very high | Very high | Very low |
| **Stream** | Medium | Very high | Very low |

The benchmark scenario is more realistic in comparison to the industrial tools, but a single benchmark does not consider the utilization during the product life cycle of a server system. The operational purpose changes over time because of growing performance demands. The server does not fulfill the requirements, and the data center manager exchanges the system or the configuration for a specific job. Another challenge is that the benchmarks are synthetic and do not show a realistic application scenario [Bor 1999, WWF et al. 2003, ERK 2006, BJ 2007, GSK et al. 2009, SLU 2010]. In addition, the application behavior is unpredictable. The identification of the key factors is crucial for the various aspects across the different abstraction levels and hardware configurations. On the other hand, a large deviation between various applications exists, which is a complex research question. Nevertheless, simulation supports the expected workload, which helps the energy-efficient characterization in the design stage of the server system. A vendor designs a server under a variety of workloads, which changes continuously over time. A challenge stated by [RF 2009] is the interaction prediction that is based only upon the resource utilization. At the same time, our challenge is the expected workload of the system because customer knowledge is a critical factor and differs between the various data center levels. We abstract the amount of time needed for a job using the actual resource utilization levels to avoid the necessity of deep application details.

If two separate tasks finish in one minute and each use the processor at a different level, what happens to the execution time or utilization level, when the processor runs both jobs in parallel? The process interaction may lead to waiting periods or performance losses. The workload characteristics influence the power- and thermal-management techniques. The authors of [Han 2007] argue, “One of the most effective techniques, DVFS, is particularly sensitive to workload memory patterns because altering the clock frequency changes the relative speed between the core processor and off-chip main memory. Compute-bound workloads are more sensitive to frequency change than memory-bound workloads.” As a result, the model has to distinguish between the utilization levels of each component for the entire system. The utilization-based server system characteristics are an advantage of the generic vendor management techniques.

Furthermore, an academic workload maps the concrete application behavior observed in an actual system, which analyzes and tests various management techniques, for instance. On the other hand, a simulation model has a predefined workload, environment, or behavior, which shows real observations. The traces also differ from each other and are not exactly at the time scale of seconds. For instance, within twenty iterations, the execution time varies approximately 1.5 minutes, whereby the average time is about 70 minutes: “Software execution also varies for the same benchmarks from run to run [Han 2007].”

**Aim #1:**

We recommend a utilization-based approach, which is compatible with the commercial tools, but also has to include low-level observations to distinguish internal structures and behavior. The benefit is the sub-characterization of each component due to the real observation level. For instance, either the processor is in an active mode like a certain p-state, or we have to describe a deeper detail, such as the frequency and voltage pair of the working conditions. Therefore, the model should include inter- and intra-component communication that refers to academic approaches. The benefit is increased accuracy because of the flexible level of detail.

A utilization pattern has to provide the scalability and flexibility of various applications. Furthermore, the approach should support customer-specific scenarios, which are more realistic in comparison to the processor-bounded method. When component interconnection is part of the simulation model, the workload abstraction may behave like several benchmarks or applications. In addition, the complete server system simulation is intended to consider recent scenarios.

A workload criterion is the opportunity to handle flexible resource utilization levels. The customer can create a specific and realistic workload scenario. In advance, our model is supposed to offer various predefined utilization levels because of the commercial compatibility.

## 4.2 Server System Characterization

The server system model characterizes the hardware precisely, which has the greatest part of the desired aspect. In the case of power analysis, the memory and processor dissipate the largest power within the server system [ERK 2006, GX 2010, THS 2010, Ste 2012, BJ 2012]. On the contrary, the network and I/O devices are key factors for the physical domain within an embedded system. Academic approaches characterize the complete system in such detail that they consider every instruction and cycle at the physical, component, or system domain in a single as well as a separated view. Each approach characterizes their component and finally the system at a specific abstraction level for their own. Scientists cannot reuse the proposed models because they are not able to map the component characteristics to their existing approaches at diverse hierarchy levels. For instance, a brand-new processor QPI speed needs a model adjustment, which commonly results in a new model because of missing compatibility and completely different component behavior. A generic model offers a robust base for the various abstraction levels. To our actual knowledge, there is no complete server system simulator in existence that considers the flexibility and scalability of diverse software and hardware configurations to reduce the modeling effort of the individual explorations [RHW et al. 1995, Her 1998, BTM 2000, ALE 2002, GSI et al. 2002, MHW 2002, MCE et al. 2002, CDS 2003, WWF et al. 2003, HSW et al. 2004, WWF et al. 2006, Lan 2007, CDS 2007, RL 2007]. A higher system flexibility and complexity may result in more computations, longer simulation time, and cut the performance because of limited simulation memory.

We found that the models do not distinguish between static and dynamic settings for the hardware and software characterization. The customers choose a hardware configuration that automatically assumes the fixed behavior predefined by the architecture or structure of the component. The authors of [HRR et al. 2007] found that five nominally identical processors have a ten percent difference in total power consumption. The components have their characteristics because of the variations within the manufacturing process. A challenge is the chip variability [Han 2007]. The industrial tools do not consider software-based settings and their influence on power consumption. An example is the memory module mode, which either refreshes the module at regular intervals, or does a series of refresh cycles. The customers select the mode only in the case of the shipped hardware, but are not able to change the settings in the configuration tool. Another example is the *Enhanced Intel SpeedStep Technology (EIST)*[172] that is configurable in the BIOS/UEFI to enable the power management. In the case of the processor, the critical power-relevant parameters are the processor speed (device scaling characteristics), core quantities, the number of active cores, and the processor core interconnections as described in Section 3.4.2. The full-system simulation tools, described in Section 3.9.1, do not support customer intervention via changing component characteristics or the environmental conditions during the simulation time. The counter-based models *Mantis* and *Joule Watcher* use the static system's observations to simulate the system [ERK 2006, Bel 2000]. *SimOS*, *SoftWatt*, and *Simics* consider a software profile at a fixed hardware configuration [RHW et al. 1995, GSI et al. 2002, MCE et al. 2002]. On the contrary, the dynamic full-system simulators *SimFlex*, *SimWattch*, and *BladeSim* distinguish between various components or physical chips within the configuration [HSW et al. 2004, CDS 2003, RL 2007]. Nevertheless, the customer cannot influence the characteristics or disturbance impacts, such as the ambient temperature. The predefined operating system and software determine the corresponding timings of the simulation model.

[172] EIST: the *Intel* processor-based power management option, *AMD's Cool 'n' Quiet™*

**Aim #2:**

We need a flexible characterization of the software and hardware to support various observations at different abstraction levels. In addition, for our simulation model we require the support of hardware-based offline settings, such as the BIOS/UEFI configuration. The model is intended to abstract the architecture and software dependencies as much as possible without losing the accuracy. A challenge is the abstraction level of the complete server system complexity, including the configuration adaptability. The abstraction level should be low enough to support architectural and structural changes at the physical domain. At the same time, a high abstraction level intends to ensure the flexibility and scalability. We recommend a component-based approach to offer upper and lower abstraction opportunities. Beneficially, the components can be defined independently of each other. As described above, the processor includes several critical parameters at all abstraction levels. Of course, the simulation model has to include dynamic approaches concerning management techniques, such as DVFS. Therefore, multiple design hierarchies are supposed to easily map the various component behaviors.

The supported component types, the device quantities, the flexible adjustment (ambient temperature) during the simulation, and the amount of statics as well as dynamic hardware and software settings are part of the characterization criteria.

## 4.3 Complete Server System Simulation

A complete server system simulation helps the industry to precisely estimate the energy efficiency and especially the power consumption of the server. The authors of [CPI et al. 2009] state "…, we need to observe systems to see how they perform in various situations." A simulation system offers the opportunity when no real hardware or a prototype is available, which reduces the costs and risks across the industry. The server system simulators model the software and hardware of the entire system, including the processor, memory, disk, network, or other devices. In the academic research, the complete system simulations use timing and behavior models at the instruction-based and cycle-accurate level, as described in Section 3.4.1. The white-box and gray-box approaches differ in their accuracy and simulation speed. The instruction-level approaches are more accurate in comparison to the gray-box approaches, but are very slow at the same time. An exact approach wastes modeling time and related costs when the share in the total power is negligible. Thus, the decision on which component and which abstraction level to use is also a key challenge.

The complete system simulators use operating system data. The OS architecture and runtime events limit the simulation model because the methods require a simulator adaption when the structure changes. It is a time-consuming process. Another challenge is that we do not know all the critical events for accurate power calculation, which also differs for various manufacturing processes of the physical domain. On the other side, if the aim is an analysis of the OS power consumption, a time-accurate, access-driven, and power-aware simulation is necessary [LJ

2003]. *SimOS* [RHW et al. 1995] includes the operating system, on a time scale of tens of milliseconds, and the corresponding software that executes complete and unmodified binary workloads. Most of the simulation approaches use instructions and cycles instead of resource utilization to consider predefined applications. Furthermore, simulation needs an explicit specification of the operating system and the executed software.

The academic approaches try "…, to explain phenomenon experienced through observations [CPI et al. 2009]" across the complex systems with a fixed target. The analytical methods have an error rate less than ten percent, but are not applicable. The simulation-based, software profile, or hardware measurement approaches are not universally valid for server systems, because of the mass of hardware and software configurations across existent and unspecific systems. The workload variation (software) with heterogeneous characteristics and the component manufacturing variation (hardware) in real systems are challenges for generic management techniques and optimization strategies, because of unexpected behavior [Han 2007].

**Aim #3:**

A flexible, entire server system simulation is intended to combine the accuracy of the academic approaches based upon some higher abstraction level. For being valuable, our model needs multiple model hierarchies, from the system to the physical domain, to include the impacts of the technology designs, generations of architectures, as well as accurate predictions of particular resources. A complete system model should provide the integration of academic approaches or results with single as well as decoupled components. The model should offer the opportunity to use white-box, gray-box, and black-box models in one simulation model. Furthermore, we intend to explore new models that cross discipline boundaries to understand the fundamental limitations and properties of power, energy, and thermal behavior. The temperature-aware workload scheduling and the energy efficiency analysis [HS 2007, MNR 2007, Han 2007, JVG 2010] show the benefits of combining the solution spaces. We should achieve a better efficiency across the overall hierarchy levels. The model should be independent of the operating system and device events. A correct abstract simulation model provides the entire system complexity to capture the most important factors, but at the same time is simple enough to cover the system behavior. In addition, the simulation model has to consider the customer and vendor requirements to support different product life cycle stages.

We use the following modeling and simulation criteria to check the recommended solution. The error rate (between measurement and calculation) is less than ten percent of the entire system. The approach has to simulate three different server generations[173]. The operating system has to be a factor for the calculation algorithm. The simulation is intended to support different product life cycle stages, which we determine by the level of component abstractions, the amount of components, and the availability that the customer can adjust the simulation.

## 4.4 Worst-Case Power Assumption

The common commercial vendors [Dell Inc., IBM Corporation, Hewlett-Packard Development Company, L.P., Fujitsu Limited] and the authors of the academic approaches [BHS 1998, ERK 2006, Riv 2008, BC 2010] accumulate the sum of all peak power values associated with each component within the system. The constant additive approaches use the worst-case scenario *“the full utilization of all resources at the same time,”* which leads to be over-provisioning. When unused resources dynamically or autonomously shut down, the worst-case assumption is invalid. Furthermore, the components execute some instructions and interact between other components. The communication needs working time and this is the reason, why the workload does not fully use all resources at the same time. A challenge is the unknown system structure, which influences the component communication.

In addition, the vendors estimate extra overhead to ensure compliance with product safety standards or future demands and call it nameplate power[174]. Significant differences between the power values occur, if we compare the nameplate power with the actual power. The authors of [New 2008] state that the nameplate power $(700W)$ is the double of the realistic peak power $(350W)$ for their chosen server example[175]. The idle power of the server system is about 200 watts. The server will never utilize the power supply over 50 percent, when vendors use the nameplate power to size the PSU. “…, Many OEMs standardize their power supplies on a smaller number of PSUs, which can result in substantial gaps between nameplate values and actual power consumed for a given piece of IT equipment, [BBJ et al. 2009].” The same authors also mention that “When discussing server power draw and cooling loads, nameplate is frequently used (albeit incorrectly) to describe the value from the server data sheet for the power supply (generally the output of the power supply listed in watts).”

In the literature of [DSC 2006], the authors state, “For the blade system, the nameplate power rating overestimates the power by almost $50\%$, and misestimates the importance of various components.” The nameplate power is substantially larger than the peak power of the entire system, which results in inefficiency, extra capital expenditures, or operational costs. A problem is that industry, especially the data center manager, generally accepts the nameplate

[173] Three generations: the previous, the recent, and the future
[174] Nameplate power: technically correct – system rating label [BBJ et al. 2009]
[175] Server configuration: x86 architecture, 1U rack server, two processors, and two hard disk drives

power to interpret incoming energy requirements. On the one side, the data center requires the electrical data, such as the wiring method and PSU quantities. On the other side, the server has to fulfill the customer-specific requirements at a particular place. Unfortunately, the data center architects still often use the nameplate power in the design, planning, and deployment stage [BBJ et al. 2009]. In addition, the power supply is not server-specific and does not change because of the changing quantities of the components. The nameplate power is independent of whether the component is mounted or not within the limits of the maximal system configuration.

**Aim #4:**

“In general, the electrical loading should be sized based on IT equipment peak measured or maximum measured power consumption levels and not nameplate values, [BBJ et al. 2009].” Therefore, we have to calculate the total power consumption independent from the worst-case scenario or create a more realistic one. The server system’s PSU sizing should use the power calculation to avoid over-provisioning.

We should consider the interdependencies between the components in the complete server system to show the real power consumption to the component utilization levels. The *Simics* or *TFSim* approach is not aware of the component interaction [MCE et al. 2002, MHW 2002]. In addition, we have to cover the non-linear behavior when we take the communication and interaction between the components into account. The authors of [ERK 2006, RRK 2008, Riv 2008] only set up fixed coefficients to handle server-specific characteristics for a predefined workload and configuration. Our model should distinguish between different inner operating modes to cover the static and dynamic component-based power with respect to the micro-architectural structures. A benefit is that we can differentiate between various states within each component separately. For instance, our model has to check if the memory is in refresh mode or executes a read/write operation. The system power is expected to rely on the individual resource utilization and their characteristics.

A criterion for exact solutions is the difference between the calculated power and the nameplate power. The simulation-based results are comparable to the measurements. The over-prediction should be less than ten percent.

## 4.5 Energy Efficiency for Peak Performance

“Most benchmarks on the basis of which systems are designed are typically structured to stress worst-case performance workloads irrespective of how the system is likely to be used in practice [Ran 2010].” Therefore, the vendors design the server system’s energy efficiency for peak performance under maximal available workload. “Data center servers usually operate far below peak utilization, which creates inefficiencies [BH 2007]”. The customer does not fully utilize a productive SVN server at every moment during a day or a week. The authors of [KFK 2008] report, that the average server utilization is less than ten percent and always lower than

50 percent in a data center. The data center manager forces the low-intensity phase (under-utilization) to have enough performance in the case of the peak capacity demand. As a result, the servers are non-utilized. In addition, the Energy Star[176] defines the typical server power as a weighted function of the idle, sleep, and standby power consumption to consider the realistic workload [Riv 2008].

Additionally, the operating point of a server system is only a local optimization for recent management techniques or optimization strategies. The vendor does not optimize the server system for under-utilization, which wastes operational expenditure in the data center. As stated in the previous chapter, the power supply efficiency is most suitable for high-intensity phases. The short-term results clarify particular moment-based problems that are not sustainable solutions. Future systems need other approaches because it is improper in the case of changing constraints or environmental conditions [CPI et al. 2009]. Secondly, the techniques are aware of the entire performance range, beginning at the low-intensity up to high-intensity phase.

**Aim #5:**

We recommend different utilization scenarios to consider under-utilization, supporting the idle, sleep, or standby mode. On the contrary, the maximal workload is not expected to be the full utilization of all resources. The authors of [ERK 2006] categorize the *SPECint* and *SPECfp* benchmark into a processor-bounded workload with high intensity. In contrast, the *stream* benchmark has a medium utilization of the processor. Beneficially, the flexible utilization-based approach has to support diverse workloads and should reproduce a realistic observation. Furthermore, the customer should influence the resource utilization at simulation time, which enables sustainable power levels. The energy efficiency characteristics are intended to show the overall behavior through different resource utilization levels. We expect to analyze critical utilization levels, such as $0\%$, $20\%$, $50\%$, $80\%$, and $100\%$, and the effect of each component. We supposed to estimate the energy efficiency on a more realistic use case, not only for peak performance.

Our model is intended to combine the thermal, power, and performance views, but we should decouple each component for more flexibility. The authors of [MSB et al. 2005] already use the same approach, but focus on the diverse memory hierarchies. The method in [TSW 2009] considers process interactions concerning the operational time spent for a process. We should exchange the data across the different layers to influence the relationships between the various aspects. For instance, higher utilization levels result in larger power consumption and better performance, but at certain points only the power value increases. The performance reaches its maximum because of limited resources or bottlenecks. The simulation approach has to support a specific behavior characterization at every abstraction level.

[176] Energy Star: https://www.energystar.gov/

The simulation offers the opportunity to optimize the system at various utilization levels, especially at under-utilization. We should be able to change the component status during simulation. Furthermore, the simulation has to provide a relative statement about the server's energy efficiency.

## 4.6 Measurement Effort

All academic system simulations have a specific research question. For instance, the approaches use different architectures of instruction sets to offer an adequate and precise model. If we change from a 32-bit architecture towards a 64-bit architecture[177], the system behaves differently and, ultimately, the interpretation of the instructions and power consumption is incorrect. A complete measurement of a server system takes several days, weeks, or months in industry. At first, the calibration phase (usually three intervals) determines the largest performance or power of the entire system under a predefined workload. A new hardware configuration requires an extra calibration of the system, which expands the time and costs. The measurement effort increases exponentially by the configuration flexibility [Fuj 2012]. Secondly, the developer measures many systems with various configurations to get correct power values of the components. A power measurement for a single component within the system under test (SUT) takes several days because of the standardized technology. The environmental laboratory ensures the prescribed controlled conditions, such as the ambient temperature at $40°C$, to reduce negative external effects and ensure comparability. A certified analyzer measures the AC power dissipation during the execution of a compliant benchmark. Hereinafter, the vendor executes the benchmark multiple times to achieve the power consumption of the system or components. Finally, the developer validates and verifies the results for each variation, which requires additional time.

The authors of [HRR et al. 2007] state that the average power varies up to ten percent during the measurement process. The academic approaches characterize the complete system in such detail that they consider every instruction and cycles, which generates a huge measurement effort in comparison to the upper abstraction level and have an increasing error rate. If industry has to measure the system at each cycle-accurate level, the time to get the power values increases exponentially.

**Aim #6:**

A flexible simulation model is intended to reduce the measurement effort, when the model combines the industrial and academic approaches. The benefits are the workload and configuration variation at a high abstraction level. On the contrary, the academic advantages are the exact algorithm at a lower level. Spreadsheet data, observations, statistical results, or customer-specific intellectual properties (IP) support the reduction of the measurements because the model may vary the server configuration in a short time and is intended to

[177] Architecture change: results in different routines for allocating and addressing the memory

simulate the entire system. The customer may change the data from a lower level to an upper level of the abstracted server system to specify the component behavior. This avoids extra measurements for a new configuration or other environmental conditions. Compared to the measurement time, the simulation time is not a critical factor. We supposed to automatically confirm the results and check it via a few separate measurements.

## 4.7 Prediction of Future and Uncertain Systems

In industry, the product life cycle never stops. System development is an iterative process, whereby the vendor improves each next generation by applying gained experiences. Each release includes novel functionality to satisfy the latest customer requirements. Therefore, the vendors have to measure the new server systems to integrate precise data within the joint configuration and order tool. The authors of [CPI et al. 2009] state, "We need to observe systems to see how they perform in various situations. This includes not only current commercially available systems, but systems purposely constructed as prototypes of new technologies." A challenge is that the necessary data for an exact model is not available, such as concrete instruction sets or operations. Furthermore, due to missing information the vendor cannot answer many open questions and values. The power consumption, especially of novel generations, is not predictable and is unknown because no material- specification assignment exists.

In addition, the hardware metrics constantly change from one generation to the next one, which means the performance counters are altered, added, or removed [Ben 2010]. The component-based power key factor relies on its architecture, structural, or functional model [BLR et al. 2005]. If a model is precisely as much as possible, it is invalid for novel generations. An upper-level abstraction results in high error rates, but on the contrary, is reusable for future components. The prediction for next-generation servers is an open research question, because the critical power factors are not available yet. Recent academic approaches do not assume the states, transitions, or behavior for future systems because of the complexity and missing data.

**Aim #7:**

Our algorithm is supposed to use the vendor experiences, spreadsheets, heuristics, and statistics to estimate the future systems on the basis of the earlier observations at every product phase. For instance, the first memory module generation $DDR-200$ offers half the transfer rate $(1.6MB/s)$ in comparison to the next-generation $DDR2-400$ $(3.2MB/s)$, which is furthermore, half of the $DDR3-800$ transfer rate $(6.4MB/s)$. Moore and Koomey stated the observed scaling and performance trends at certain manufacture generations of the fabrication technology, but did not investigate the configuration-specific parameters [Moo 1965, KBS et al. 2010]. The model is intended to use the generic findings, which leads to the prediction of the largest throughput of next generations. We have to predict future systems on an upper abstraction level without details about the structure or instructions.

The customer is intended to change the component characteristics to assume the future behavior of the next-generation components. The model should provide the availability to add new components and server systems because of the decoupled and hierarchical concept at various abstraction levels. The virtual prototype of a component or server system is a benefit in the early design stage. The comparison between the prediction and the state-of-the-art component is, consequently, an evaluation criterion.

## 4.8 Summary

Our aim is to improve the power calculation in the commercial tools supported by the academic approaches regarding realistic and customer-specific workload scenarios for flexible server system variations. The over-provisioning reduction is a further dissertation goal achievable by more precise worst-case power calculation and adjustment. Another aspect is the server system's energy efficiency, which is predictable at all utilization levels. If the approach decreases the measurement effort, the vendors will save time and costs. The future resource prediction supports the applicability at certain product development stages.

# 5 Multi-aspect Full-system Server Model and Optimization Concept as a Simulation-based Approach (MFSMOS)

We develop a generic and scalable concept that consists of five separate steps to support the dynamic adaptations of control algorithms, management strategies, and system characteristics. Figure 43 shows an overview of the main concept. The *externals* block $(EX)$ describes the environment and generates the stimuli for the simulation. We map the *externals* to the *characterization* block $(SY)$, where we model and characterize the server system, its components, and its chips primarily in the physical domain, using a mix of commercial and academic algorithms to create multi-aspect models. We *simulate* the entire server system using the characteristics $(SY)$ and resource utilization levels $(EX)$. The *results* block stores the simulation response, in particular the energy efficiency values, to analyze the effects of each customization and autonomous management decision. The analysis is part of the *optimization strategy* in which we adapt the simulation by changing the *characterization* or partly changing *externals*. We conclude the simulation, its results, and the corresponding optimization strategies into the full-server system simulation $(FS)$. The simulation is performed offline.

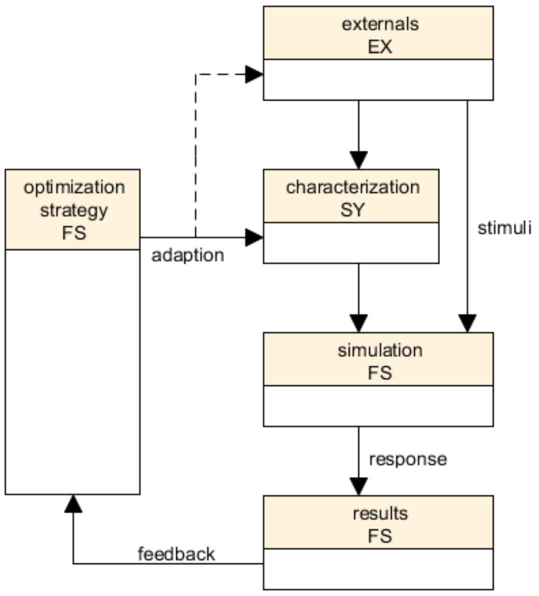


Figure 43: Overview of the five-step concept

Section 3.9 describes the state-of-the-art full-system approaches and simulators. As part of our overall system, we conceive an alternative server system simulator that is independent of the operating system. Our flexible external and internal system characterization supports any server and component vendor. We develop a holistic concept that includes the results of theoretical analyses, heuristics, measurements, and vendor-based experiences from existing systems as well as next-generation systems in the early design phases. At this stage, we are not able to characterize all components at the same level of detail. Therefore, our model handles various abstraction levels beginning with instructions, proceeding with states, and finishing with black-box descriptions where we predict the behavior on the basis of spreadsheets. We support flexible and configurable resource utilization levels to estimate customer-specific application software, which is the major simulator input. Furthermore, we model the impact of system-tailored configurations and characteristics considering the following aspects: power, performance, and temperature. The simulator includes common power management techniques, as shown in [RL 2007]. In contrast, the authors in [RL 2007] trace and store the corresponding utilization levels of an explicit application of a real system with a static configuration. In addition, we consider the external power limits and thermal limits provided by the customer. Therefore, we include various management techniques and model the related behavior. According to the simulation results, we optimize the energy efficiency by adapting the system characterization. Finally, the simulator provides ideal energy-efficient settings for the given resource utilization levels. If no single optimum is possible, the system operates at an energy-efficient corridor, characterized by a Pareto front. We describe the full-system simulations in Section 5.1 and Section 5.4.2.

In this chapter, we assign the following terms with respective notations. We define the indices $i, j, n, m, k, l$, which are always any natural number $N_0 = \{0,1,2,3, \dots\}$. In particular, an index $i$ specifies a concrete component $C_i$ or category $CS_i$. The related index $m$ identifies the maximal number of the elements in the set of $C$ or $CS$[178]. We specify an aspect $A_j$ using the index $j$, whereby $n$ defines the final element.

$$i, j, n, m, k, l \in N_0,\ N_0 = \{0,1,2,3, \dots\}, N = \{1,2,3, \dots\} \tag{5.1}$$

$$C = \{C_1, C_2, \dots, C_m\}, C_i \in C \tag{5.2}$$

$$CS = \{CS_1, CS_2, \dots, CS_m\}, CS_i \in CS \tag{5.3}$$

$$A = \{A_1, A_2, \dots, A_n\}, A_j \in A \tag{5.4}$$

We use the index $k$ and $l$ as any consecutive index of all remaining terms. For instance, lambda is a generic object that has $k$ elements, as shown in Equation (5.5). The index $l$ specifies an explicit element of the set $\Lambda$.

$$\Lambda = \{\Lambda_1, \Lambda_2, \dots, \Lambda_k\}, \Lambda_l \in \Lambda \tag{5.5}$$

In addition, we add an extra dimension, whereby the first dimension specifies the $l$-th element of $\Lambda$, and the second dimension defines the related component $C_i$. We determine each element by $\Lambda_{l_{C_i}}$ in the two-dimensional array, see Equation (5.6). In the case that we set $l = 2$ to a fixed value, $\Lambda_2$ looks like Equation (5.7), but includes an extra system-wide $\Lambda_{2_{SY}}$ element.

178 Component and categories: We state further details of the context in Section 5.3.2.1.

$$\Lambda = \left\{\Lambda_{1_{C_1}}, \Lambda_{2_{C_2}}, \dots, \Lambda_{k_{C_m}}\right\}, \Lambda_{l_{C_i}} \in \Lambda \tag{5.6}$$

$$\Lambda_2 = \left\{\Lambda_{2_{SY}}, \Lambda_{2_{C_1}}, \Lambda_{2_{C_2}}, \dots, \Lambda_{2_{C_m}}\right\}, \Lambda_{2_{C_i}} \in \Lambda_2 \tag{5.7}$$

The following section describes aspect-based component models as the basis of our full-server system simulation.

## 5.1 Aspect-based Component Models of the Full-Server System Simulation

We define a dynamic and deterministic simulation model using time-continuous and value-continuous stimuli. The *externals* $EX$ build the stimuli for the simulation model, which we convert into a time-discrete and a value-discrete workload. We need steady states because the simulation framework calls our calculation methods at each simulation step. We calculate the energy efficiency of the server, which depends upon the actual system configuration. Therefore, we characterize the system and its system-board components $C_i$ to define the inner behavior. At first, we specify each component separately in order to define the aspect-related functions. Afterwards, we include the relations between the aspects of a single component. Finally, we define component's relations of the entire system behavior. Table 24 and Table 25 list the symbols and definitions for the aspect-based component models within our simulation.

Table 24: Nomenclature – aspect-based component models $(I)$

| Nomenclature | Meaning | Nomenclature | Meaning |
|---|---|---|---|
| $\boldsymbol{A}$ | Aspect | $\boldsymbol{i, j, n, m, k, l}$ | Index |
| $\boldsymbol{C}$ | Component | $\boldsymbol{N_0}$ | Any natural number $N_0 = \{0,1,2,3, \dots\}$ |
| $\boldsymbol{CS}$ | System-board category ( $\equiv$ components) | $\boldsymbol{\theta}$ $\boldsymbol{\theta_R, \theta_C, \theta_S}$ | Configuration tree (HW, SW) released, customer, system-compatible |
| $\boldsymbol{EX}$ | Externals | $\boldsymbol{proc}$ | $Processor$ |
| $\boldsymbol{SY}$ | System | $\boldsymbol{mem}$ | $Memory$ |
| $\boldsymbol{FS}$ | Full-system simulation and optimization | $\boldsymbol{io}$ | $Input/output$ |
| $\boldsymbol{AC}$ | Architecture | $\boldsymbol{oth}$ | $Others$ |
| $\boldsymbol{CC}$ | Connectors | $\boldsymbol{MAS_C}$ | Aspect-based models per component |
| $\boldsymbol{EE}$ | Energy efficiency | $\boldsymbol{A_{j_{C_i}}}$ | Element in matrix $MAS_C$ |
| $\boldsymbol{PO}$ | Power | $\boldsymbol{F_{A_{j_{C_i}}}}$ | Functional description of $A_{j_{C_i}}$ |
| $\boldsymbol{PE}$ | Performance | $\boldsymbol{F(x)}$ | Objective functions with decision variables $x$ |

Table 25: Nomenclature – aspect-based component models $(II)$

| Nomenclature | Meaning |
|---|---|
| $\boldsymbol{TH}$ | Thermal |
| $\boldsymbol{DY_s}$ | Dynamic behavior of the system (characteristics) |
| $\boldsymbol{ST_s}$ | Static characteristics |
| $\boldsymbol{R^{A_k}_{A_l}}$ | Relation between the aspects $(A_k, A_l)$ |
| $\boldsymbol{R_{A_{j_{C_i}}}}$ | Aspect-related relevance for component $i$ and aspect $j$ |
| $\boldsymbol{RR^{A_{k_{rel}}}_{A_{j_{rel}}}}$ $\boldsymbol{{}_{min}RR^{A_{k_{rel}}}_{A_{j_{rel}}}}$ $\boldsymbol{{}_{max}RR^{A_{k_{rel}}}_{A_{j_{rel}}}}$ | Interval limits of relation $R^{A_k}_{A_l}$ |
| | |

| Nomenclature | Meaning |
|---|---|
| $\boldsymbol{x}$ | Decision variables |
| $\boldsymbol{G(x), G^C_1(x)}$ | Constraints of $F(x)$ |
| $\boldsymbol{AX, AY}$ | Aspect-based models |
| $\boldsymbol{R_A, R_{A_C}}$ | Aspect-based relations |
| $\boldsymbol{{}_{A_j}R^{t_{k+1}}_{t_k}}$ | Impact of $A_j$ at time step $t_{k+1}$ |
| $\boldsymbol{BE_C}$ | Component behavioral model |
| $\boldsymbol{R_{BE}}$ | Relations between the component-specific behavior models |

This section defines the formal description of the aspect-based component models which we want to cover in the simulation. We consider the set of aspects $A$ and the set of components $C$, see Equations (5.8) and (5.9).

$$A = \{A_1, A_2, \dots, A_n\}, A_j \in A \tag{5.8}$$

$$C = \{C_1, C_2, \dots, C_m\}, C_i \in C \tag{5.9}$$

In our concept, we generate models for all combinations of the components $C_i \in C$ and aspects $A_j \in A$ that we consider in the simulation, whereby $i, j, m, n$ are any natural numbers $i, j, m, n \in N_0, N_0 = \{0, 1, 2, 3, \dots\}$: i.e., they build the cross product $A \times C$, see Equation (5.10). We concentrate upon the behavioral model of each component $C_i$ in the system.

$$MAS_C = A \times C = \begin{pmatrix} (A_1, C_1) & (A_1, C_2) & \dots & (A_1, C_m) \\ (A_2, C_1) & (A_2, C_2) & \cdots & (A_2, C_m) \\ \vdots & \vdots & \ddots & \vdots \\ (A_n, C_1) & (A_n, C_2) & \cdots & (A_n, C_m) \end{pmatrix} = \begin{pmatrix} \left(A_{1_{C_1}}\right) & \left(A_{1_{C_2}}\right) & \dots & \left(A_{1_{C_m}}\right) \\ \left(A_{2_{C_1}}\right) & \left(A_{2_{C_2}}\right) & \cdots & \left(A_{2_{C_m}}\right) \\ \vdots & \vdots & \ddots & \vdots \\ \left(A_{n_{C_1}}\right) & \left(A_{n_{C_2}}\right) & \cdots & \left(A_{n_{C_m}}\right) \end{pmatrix} \quad (5.10)$$

$$MAS_C = MAS_{A_{j_{C_i}}} = (A_{j_{C_i}}) \quad (5.11)$$

We define the component $C_1$ by all aspects $A = \{A_1, A_2, \dots, A_n\}$ and the aspect $A_1$ for all components $C = \{C_1, C_2, \dots, C_m\}$. The row $n$ of the two-dimensional array specifies the aspect $A_n$, and the column $m$ specifies the component $C_m$, see Equation (5.12).

$$A = \left\{A_{1_{C_1}}, A_{2_{C_2}}, \dots, A_{n_{C_m}}\right\}, A_{j_{C_i}} \in A \quad (5.12)$$

In other words, each element within the matrix $MAS_C$ defines an aspect-based component model. A row vector $MAS_{A_{j_{C_i}}}$ specifies one aspect $A_j$ for all components $C_i \in C$. In contrast, a column vector $MAS_{A_{j_{C_m}}}$ specifies all aspects $A_j \in A$ for one component $C_m$.

$$MAS_{A_{j_{C_i}}} = \left(\left(A_{j_{C_1}}\right) \quad \left(A_{j_{C_2}}\right) \quad \left(A_{j_{C_m}}\right)\right) \qquad MAS_{A_{j_{C_m}}} = \begin{pmatrix} \left(A_{1_{C_m}}\right) \\ \left(A_{2_{C_m}}\right) \\ \vdots \\ \left(A_{n_{C_m}}\right) \end{pmatrix} \quad (5.13)$$

Finally, our matrix $MAS_C$ contains all component models for all aspects $A_j$. We define each element $A_{j_{C_i}}$ of the two-dimensional array $MAS_C$ in the *logical and physical* layer. Other full-system simulators, see Section 3.2.2 and Section 3.9, do not cover diverse aspects at the same time. In our concept, we address the aspects power $PO$, performance $PE$, and thermal $TH$ for all components, see Equations (5.14) and (5.15).

$$A = \{PO, PE, TH\} = \{A_1, A_2, A_3\}, A_j \in A \mid \forall C_i \quad (5.14)$$

$$MAS_{A_{C_i}} = \begin{pmatrix} PO_{C_i} \\ PE_{C_i} \\ TH_{C_i} \end{pmatrix} \quad (5.15)$$

We consider the following components: $processor$, $memory$, $input/output$, $fan$, and $others$, which we describe in Section 5.3.2.1. Equation (5.17) shows the row vector $MAS_{PO_{C_i}}$, which refers to the power models of all components $C_i$.

$$C = \{proc, mem, io, fan, oth\} \tag{5.16}$$

$$MAS_{PO_{C_i}} = (PO_{proc} \quad PO_{mem} \quad PO_{io} \quad PO_{fan} \quad PO_{oth}) \tag{5.17}$$

$$MAS_C = \begin{pmatrix} PO_{proc} & PO_{mem} & PO_{io} & PO_{fan} & PO_{oth} \\ PE_{proc} & PE_{mem} & PE_{io} & PE_{fan} & PE_{oth} \\ TH_{proc} & TH_{mem} & TH_{io} & TH_{fan} & TH_{oth} \end{pmatrix} \tag{5.18}$$

We generate the entire aspect-based component matrix $MAS_C$ and create a model for each element within the matrix. We define the calculation methods for each element $A_{j_{C_i}}$ as a functional description $F_{A_{j_{C_i}}}$, which we describe in Section 5.2.2.

We additionally consider relations $R_A$ between aspect-based models $AX$ and $AY$, which are elements within the matrix $MAS_C$. Each model $AX$ and $AY$ contain the aspects $A_j$ or $B_j$, respectively, where the number of aspects is considered to be equal. We specify each combination in the two-dimensional matrix $R_A$, see Equation (5.22).

$$R_A = AX \times AY \tag{5.19}$$

$$AX = \{A_1, A_2, \dots, A_n\}, AY = \{B_1, B_2, \dots, B_n\} \tag{5.20}$$

$$R_A = AX \times AY = \{(A_j, B_j) | A_j \in AX, B_j \in AY, j \in N_0\} \tag{5.21}$$

$$R_A = \begin{pmatrix} (A_1, B_1) & (A_1, B_2) & \dots & (A_1, B_n) \\ (A_2, B_1) & (A_2, B_2) & \cdots & (A_2, B_n) \\ \vdots & \vdots & \ddots & \vdots \\ (A_n, B_1) & (A_n, B_2) & \cdots & (A_n, B_n) \end{pmatrix} \tag{5.22}$$

The matrix contains all combinations of $AX \times AY$. As stated before, we always address the same aspects $A_j$ for all components $C_i$. Therefore, $AX$ is identical with $AY$, we replace $AY$ in $R_A$ and create a simplified version, see Equation (5.25).

$$A = AX = AY = \{A_1, A_2, \dots, A_n\} \mid \forall C_i, i \in N_0 \tag{5.23}$$

$$R_A = A \times A = \begin{pmatrix} (A_1, A_1) & (A_1, A_2) & \dots & (A_1, A_n) \\ (A_2, A_1) & (A_2, A_2) & \cdots & (A_2, A_n) \\ \vdots & \vdots & \ddots & \vdots \\ (A_n, A_1) & (A_n, A_2) & \cdots & (A_n, A_n) \end{pmatrix} \tag{5.24}$$

A relation between the same aspects $A_j$ does not exist and is negligible, see Equation (5.25). For compatibility in the tool chain, we define the relation of two identical aspects $A_j$ as value *one* within the matrix $R_A$. The relation between the two aspects $A_1$ and $A_2$ is bidirectional and interchangeable, see Equation (5.26).

$$\left(A_j, A_j\right) = 1 \tag{5.25}$$

$$(A_1, A_2) \equiv (A_2, A_1) \tag{5.26}$$

As a result, we specify a simplified matrix $R_A$, which defines all existing relations between the aspects $A_j$, again setting all values to be ignored to 1.

$$R_A = A \times A = \begin{pmatrix} 1 & (A_1, A_2) & (A_1, A_3) & \dots & (A_1, A_n) \\ 1 & 1 & (A_2, A_3) & \cdots & (A_2, A_n) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \cdots & 1 \end{pmatrix} = \begin{pmatrix} 1 & R_{A_2}^{A_1} & R_{A_3}^{A_1} & \dots & R_{A_n}^{A_1} \\ 1 & 1 & R_{A_3}^{A_2} & \cdots & R_{A_n}^{A_2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \cdots & 1 \end{pmatrix} \tag{5.27}$$

We define the element $(A_k, A_l)$ in $R_A$ as the relation $R_{A_l}^{A_k}$. We summarize the relations among the component $C_i$ by $R_{A_{C_i}}$. Finally, we collect all component relations in $R_{A_C}$.

$$R_{A_{C_i}} = \{R_{A_2}^{A_1}, R_{A_3}^{A_1}, R_{A_3}^{A_2}, \dots, R_{A_n}^{A_1}, R_{A_n}^{A_2}, R_{A_n}^{A_{n-1}}\} \tag{5.28}$$

$$R_{A_C} = \{R_{A_{C_1}}, R_{A_{C_2}}, \dots, R_{A_{C_n}}\} \tag{5.29}$$

According to the specification in (5.27), we define the relations $R_A$ for the aspects $A = \{PO, PE, TH\}$ and specify them for all components $C_i$ in $R_{A_{C_i}}$.

$$R_A = \begin{pmatrix} 1 & R_{PE}^{PO} & R_{TH}^{PO} \\ 1 & 1 & R_{TH}^{PE} \\ 1 & 1 & 1 \end{pmatrix} = \{R_{PE}^{PO}, R_{TH}^{PO}, R_{TH}^{PE}\} \tag{5.30}$$

$$R_{A_{C_i}} = \{R_{PE_{C_i}}^{PO_{C_i}}, R_{TH_{C_i}}^{PE_{C_i}}, R_{TH_{C_i}}^{PO_{C_i}}\} \tag{5.31}$$

Figure 44 illustrates a component $C_i$, its aspect-based models of the matrix $MAS_C$, and the models' corresponding relations $R_{A_{C_i}}$.

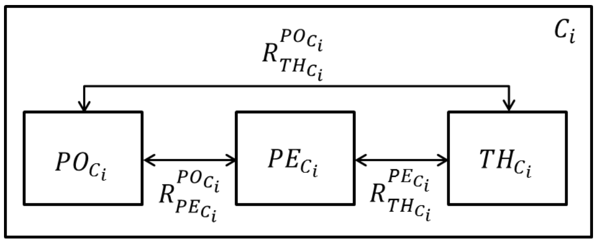

$C_i$
$R_{TH_{C_i}}^{PO_{C_i}}$
$PO_{C_i}$
$PE_{C_i}$
$TH_{C_i}$
$R_{PE_{C_i}}^{PO_{C_i}}$
$R_{TH_{C_i}}^{PE_{C_i}}$

**Figure 44: Aspect-based components and corresponding relations $(R_A)$**

In the next step, we specify the behavior model $BE_{C_i}$, whereby we include the aspect-based models in $MAS_{A_{C_i}}$ for each component $C_i$and their relations $R_{A_{C_i}}$. We define the components' behavioral level, but do not specify the description level for each domain. For instance, we specify the processor power model on the physical domain, but the memory thermal model as a black-box approach. Equation (5.33) defines the behavior model of the processor, which includes the aspect-based models $PO_{proc}, PE_{proc}, TH_{proc}$ considering the relations $R_{PE_{proc}}^{PO_{proc}}, R_{TH_{proc}}^{PE_{proc}}$, and $R_{TH_{proc}}^{PO_{proc}}$. We collect the system behavior model in $BE_C$, see Equation (5.34).

$$BE_{C_i}(MAS_{A_{C_i}}, R_{A_{C_i}}) \tag{5.32}$$

$$BE_{proc}(PO_{proc}, PE_{proc}, TH_{proc}, R_{PE_{proc}}^{PO_{proc}}, R_{TH_{proc}}^{PE_{proc}}, R_{TH_{proc}}^{PO_{proc}}) \tag{5.33}$$

$$BE_C = \{BE_{C_1}, BE_{C_2}, \dots, BE_{C_m}\} \tag{5.34}$$

The authors of [RL 2007, Riv 2008, BC 2010] use a state-based approach to specify the system and components, but do not consider the component's interaction. For each component $C_i$ we separately define a behavioral model $BE_{C_i}$, but a component can influence the behavior of another component. The processor has to wait for the memory executions and autonomously switches to the idle state, for instance. We specify the relations $R_{BE}$ between the component-specific behavior models $BE_C$ as a cross product, see Equation (5.35).

$$R_{BE} = BE_C \times BE_C \tag{5.35}$$

According to the assumptions for the aspect-based relations $R_A$, we generate a model $R_{BE}$, see Equations (5.36), (5.38), and (5.39). Our approach shall automatically generate and fill the matrix. We concentrate on the behavior of each component and integrate the consequent to the other components. We simulate the constant architecture $AC$ of our server system, because of the existing connectors $CC$ between the mounted components.

$$R_{BE_{C_1}}^{BE_{C_1}}, R_{BE_{C_2}}^{BE_{C_2}}, \dots, R_{BE_{C_n}}^{BE_{C_n}} = \{\emptyset\} \tag{5.36}$$

$$R_{BE_{C_i}}^{BE_{C_i}} = 1 \tag{5.37}$$

$$R_{BE_{C_2}}^{BE_{C_1}} \equiv R_{BE_{C_1}}^{BE_{C_2}} \tag{5.38}$$

$$R_{BE} = BE_C \times BE_C = \begin{pmatrix} 1 & R_{BE_{C_2}}^{BE_{C_1}} & R_{BE_{C_3}}^{BE_{C_1}} & \cdots & R_{BE_{C_m}}^{BE_{C_1}} \\ 1 & 1 & R_{BE_{C_3}}^{BE_{C_2}} & \cdots & R_{BE_{C_m}}^{BE_{C_2}} \\ 1 & 1 & 1 & \cdots & R_{BE_{C_m}}^{BE_{C_3}} \\ 1 & 1 & 1 & \ddots & \vdots \\ 1 & 1 & 1 & \cdots & 1 \end{pmatrix} \tag{5.39}$$

The aspect-based models in $MAS_C$, their relations $R_A$, and the behavioral relations $R_{BE}$ specify the dynamic behavior within the system $DY_S$. Herein, we define the system-specific dependencies based upon the explicit configuration tree $\theta_C$, mounted components $C_i$, and their category-specific characteristics. Finally, our system model consists of static $ST_s$ and dynamic $DY_s$ characteristics. Besides the dynamic configuration, we specify the system architecture $AC$ and the related connectors $CC$ as static characteristics $ST_s$, which are not configurable within the system, such as structural restrictions about the amount of components $C_i$ in $\theta_C$. Figure 45 shows a graphical overview of the dynamic characteristics in our system model.

$$DY_s = \{BE_C, R_{BE}\} = \{MAS_C, R_{A_C}, R_{BE}\} \tag{5.40}$$

$$ST_s = \{AC, CC\} \tag{5.41}$$

$$SY = \{ST_s, DY_s\} \tag{5.42}$$

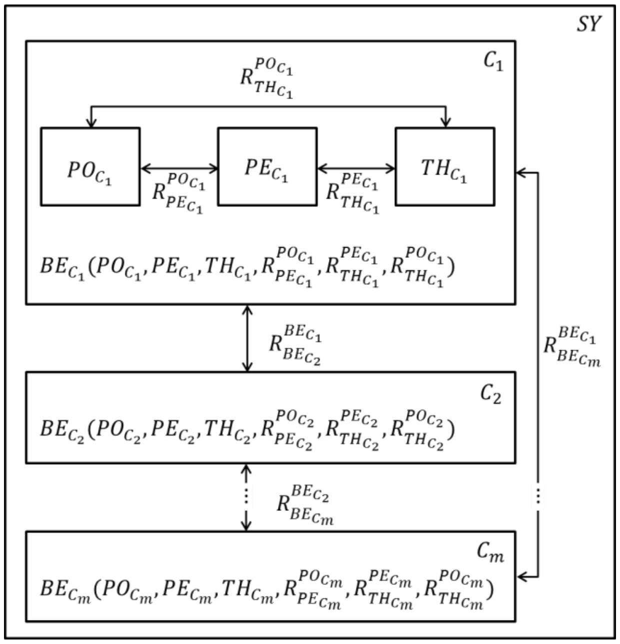

$SY$
$C_1$
$R_{TH_{C_1}}^{PO_{C_1}}$
$PO_{C_1}$
$PE_{C_1}$
$TH_{C_1}$
$R_{PE_{C_1}}^{PO_{C_1}}$
$R_{TH_{C_1}}^{PE_{C_1}}$
$BE_{C_1}(PO_{C_1}, PE_{C_1}, TH_{C_1}, R_{PE_{C_1}}^{PO_{C_1}}, R_{TH_{C_1}}^{PE_{C_1}}, R_{TH_{C_1}}^{PO_{C_1}})$
$R_{BE_{C_2}}^{BE_{C_1}}$
$R_{BE_{Cm}}^{BE_{C_1}}$
$C_2$
$BE_{C_2}(PO_{C_2}, PE_{C_2}, TH_{C_2}, R_{PE_{C_2}}^{PO_{C_2}}, R_{TH_{C_2}}^{PE_{C_2}}, R_{TH_{C_2}}^{PO_{C_2}})$
$R_{BE_{Cm}}^{BE_{C_2}}$
$C_m$
$BE_{C_m}(PO_{C_m}, PE_{C_m}, TH_{C_m}, R_{PE_{Cm}}^{PO_{Cm}}, R_{TH_{Cm}}^{PE_{Cm}}, R_{TH_{Cm}}^{PO_{Cm}})$

Figure 45: System model $(MAS_C, R_{A_C}, R_{BE})$

In conclusion, in our concept we define the aspects power $PO$, performance $PE$, and thermal $TH$ to estimate the energy efficiency $EE$. The *externals* block $EX$ defines the working conditions to the server system. The previous formal descriptions characterize the system $SY$ to calculate a set of heterogeneous aspects $A_j$, which we want to optimize.

$$[EX][SY] \rightarrow [A_j] \tag{5.43}$$

$$[EX][SY] \rightarrow [PO \quad PE \quad TH] \tag{5.44}$$

We specify the dynamic and static characteristics of the system and its components. Then we replace the system $SY$ with our models, as shown in Equation (5.46). Furthermore, we characterize the components within the server system $SY$ in order to precisely calculate each aspect $A_j$, which we require for our simulation.

$$[EX]\begin{bmatrix} ST_s \\ DY_s \end{bmatrix} \rightarrow [PO \quad PE \quad TH] \tag{5.45}$$

$$[EX]\begin{bmatrix} ST_s \\ MAS_C, R_{A_C}, R_{BE} \end{bmatrix} \rightarrow [PO \quad PE \quad TH] \tag{5.46}$$

The following sections describe various concept stages and introduce what the respective blocks have to provide. We present existing approaches and argue why the concepts are not fully applicable for the server system simulation. Afterwards, we emphasize our contributions and describe which approaches we use or adapt.

## 5.2 Server System Configuration and Characterization $SY = \{\theta, \theta_{TS}, \theta_C, \theta_{CS}, \beta, \delta, \gamma, \upsilon, \chi\}$

This section describes the fundamental principles to simulate an entire server system. The explicit server configuration specifies the architecture and the respective components, which we further characterize to calculate the concrete aspects. We specify a hierarchical approach, similar to [GFN et al. 2006], that provides the scalability to define the system from upper to lower abstraction levels. In our approach, we conceive a flexible model that decouples the layers *configuration*, *logical and physical,* and *process and control*, which we define separately from each other, allowing to support independent descriptions of the diverse domain-specific characteristics. The authors of [Che 2006] analyze an application referring to the components *processor*, *cache*, *memory*, and *peripheral*. The authors propose a model of each component and calculate the total power consumption. We consider multi-aspects so that we cannot apply the approach directly. We develop a generic model and use the utilization levels instead of the instruction sets or particular memory traces.

The *configuration* layer defines the customer-specific system configuration. The *configuration* layer describes the physical system from the structural perspective, which considers the maximal amount of possible mountable system-board components $C_i$. Herein, we cover the system architecture, design, and structure of the entire server system. We model the system architecture encapsulated of the components, which supports multiple server generations without creating a completely new model. The *configuration* layer supports the interactions with our optimization strategies, which alters the specific configurations of the server system to find an ideal energy-efficient solution. We concentrate upon the entire system and differentiate into the possible supported components and the simulation configuration, which is usually specified by the customer in the commercial tools. In our simulation model, we require a *Fujitsu System Architect* file[179], which specifies the customer-specific server system. In principle, we parse the file[180] to define the server configuration and its mounted components. Further characterization is done through a generic configuration tree that considers the explicit system configuration and its particular specification. We differentiate into the configuration and characterization of the server system and its components to define the aspect-based models. We specify the calculation methods of each component, whereby we access these functions within our simulation model. Therefore, we analyze the components within the characterization layer to find the significant energy efficiency characteristics of each component $C_i$ in every aspect $A_j$, but in an abstract manner. We specify the functional models concerning the thermal, power, and performance aspects of each component in the *logical and physical* configuration layer to calculate the energy efficiency at every simulated time $t_k$. We define the diverse levels of the component details flexibly based upon our knowledge of the corresponding accuracy level and data, see Section 5.2.2. In addition, the characterization layer considers the management techniques $\gamma$, such as dynamic voltage frequency scaling (DVFS) and dynamic thermal management (DTM). We distribute the workload (utilization levels) towards the mounted components and define the communication $\delta$ between the components $C_i$ in the *process and control* layer. Our *process and control* layer includes the calculation methods based upon the *characterization* layer to provide the necessary simulation data. We manage the system behavior and consider the internal system constraints $\upsilon$ as well as simulation constraints $\chi$, see Section 5.2.3. All layers together generally characterize the server system and its behavior. Figure 46 shows a brief overview of the server system characterization and its layers, which we outline in the next sections.

[179] *Fujitsu System Architect* file: proprietary format, http://configurator.ts.fujitsu.com/public/

[180] File support: We have to extend the interface and parsing algorithm, which supports server configuration files of the other commercial tools.

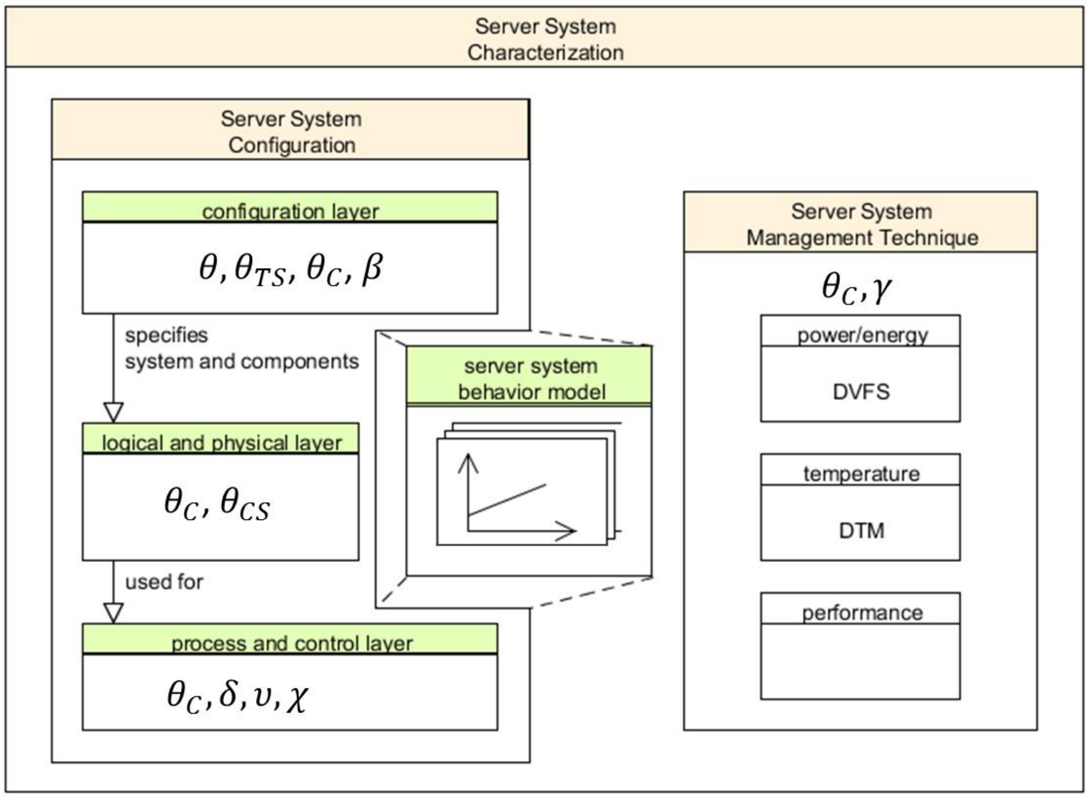

Server System Characterization
Server System Configuration
configuration layer
$\theta, \theta_{TS}, \theta_C, \beta$
specifies system and components
logical and physical layer
$\theta_C, \theta_{CS}$
used for
process and control layer
$\theta_C, \delta, \upsilon, \chi$
server system behavior model
Server System Management Technique
$\theta_C, \gamma$
power/energy
DVFS
temperature
DTM
performance

Figure 46: Server system characterization

### 5.2.1 Constituents of the Simulation Model

Our simulation model has a data layer that stores the server system configurations and characterizations in a centralized database. The aspect-based calculation methods $F_{A_{j_{C_i}}}$ in $MAS_C$ use the extended component information[181] coming from this database. We define the equations in the *logical and physical* layer, which we separate from the data layer to support diverse abstraction levels. We provide access to individually configurable data within the database to enable the usage of our models across multiple server generations over several years. We recognize two classes of users: the system administrator and the customer. The system administrator is the expert who configures the weight coefficients of the calculation methods, for instance, updates the database and maintains the simulation model. The customer specifies the external input parameters of the simulation model to his or her need. The most critical simulation input parameters are the customer-specific system configuration and the corresponding workload. We alter the server configuration as part of our optimization strategy because, e.g., two memory modules may provide a better performance and consume less power in comparison to one module that has the same technical specification and total

[181] Extended component information: customer-specific system configuration does not provide the certain details of a component, such as the memory fabrication size

capacity. The amount of components within the system influences the energy efficiency. On the other hand, a current processor from today may be the bottleneck of the entire server system. If we choose a faster processor or a second one, the memory will become the bottleneck. We define a generic configuration tree $\theta$ that specifies the actual physical system from the structural perspective concerning the maximal amount of possible mountable system-board components $C_i$. The design and architectural descriptions include the topology, hierarchy, component types, and connections. The configuration tree considers all possible server system configurations independent of the vendor, generation, structure, or hardware restrictions. We concentrate upon an exact server system configuration within one simulation run, which we consider as the initial configuration of the simulation $\theta_C$. We separate the system architecture model to abstract the wide range of components from an explicit server generation. We reuse the component models for multiple server generations and avoid additional effort instead of creating a completely new model for each generation.

**Server System Configuration Tree $\theta$ and Components $C_i$**

The sub-component tree of a server, as addressed in [GFN et al. 2006], builds the base of our modular and hierarchical concept. The authors include the dynamic and static properties within the tree, but do not care about the architectural dependencies, such as generation, family, or revision to support heterogeneous components. Our aim is to depict a wide range of customer configurations $\theta_C$ and optimize the system's energy efficiency. We adjust and enhance the component tree of [GFN et al. 2006] to abstract the hardware components from their explicit technical specification and extend their model by a flexible characterization. We define any possible system configuration $\theta$, its architecture, and resources, but avoid the redundant data in comparison with the configuration tree in [GFN et al. 2006]. We conceive a general configuration tree[182] by a flexible amount of subtypes $SU_k$ and sub-subtypes $SU_{kl}$, which are the vertices, as formally defined in Section 2.2.6. We define the server system configuration as a tree to extend and integrate dynamic new resources by adding an extra subtype[183]. We can reduce the complexity of our simulation by deleting a subtype of the tree, which supports the flexibility when we want to reorder or reorganize the components based upon the updated findings of the measurements or the next-generation architectures. We can easily apply search algorithms, annotate the tree elements, and depict dependencies. The edges between the vertices describe the "consist of" relation. A subtype may consist of additional (multiple) sub-subtypes, for instance, or does not have any children, which are called leaves. We define the subtypes and sub-subtypes as flexible, dynamic structures at each level within the configuration tree, which are expandable for next-generation systems. The edges in the technical specification tree $\theta_{TS}$ are an exception because we use them as an "is a"

[182] General configuration tree: has no maximal number of possible subtypes (degree), and every subtype can have any number of subtypes independent of each other
[183] Subtype: vertex (node) in the tree

relation. The general configuration tree includes all supported components and server configurations within our simulation model. Therefore, we analyze the *Fujitsu System Architect* database to generate the all-encompassing configuration tree considering the several configurations and characterizations that are possible to specify an entire server system. The configuration tree uses a centralized database that considers the various components independent of the vendor.

We distinguish between the released configuration $\theta_R$, the customer configuration $\theta_C$, and any system-compatible configuration $\theta_S$ within our configuration tree $\theta$ with $\theta_C \subseteq \theta_R \subseteq \theta_S \subseteq \theta$. A customer configuration is always a released and system-compatible configuration. The vendor restricts the released hardware configuration because of the compatibility to predecessor generations or vendor-specific constraints. The customer configuration $\theta_C$ is our initial configuration and serves as simulation input. One of our optimization strategies is the alternation of the hardware configuration. We exchange the components $C_i$ within $\theta_C$, whereby $C_i$ is system-compatible but not necessarily a released[184] server configuration $\theta_R$. This supports our flexible concept in order to simulate various hardware configurations and their corresponding characteristics. As stated in the background section, we define our configuration tree $\theta$ using the following subtypes and sub-subtypes for a rack-mounted server system $S$, which is the root type $TY_0$, see Figure 18:

- Software $SW$
- Hardware $SH$
    - Components $HC$
        - Add-in $CA$
        - On-board $CO$
        - System-board $CS$ (components $C$)
    - Connectors $HO$
    - Power supply $HP$

We address the software settings $SW$ in the *externals* block $EX$, which we include as settings $\xi$. Our workload model covers various applications, see Section 5.3.2.2. We consider the power supply $HP$ within the total power calculation. We differentiate the connectors $HO$ into virtual connectors, such as associated network uplink, and electrical connectors $CC$. The electrical connectors[185] enable the communication and limit the maximal performance because of the resources' throughput and bandwidth. We do not cover add-in components $CA$, because of their wide variety and in order to reduce complexity. In addition, we abstract the on-board mounted components $CO$, which we denote as main board-specific base power. Usually, we

[184] Non-released servers: operational servers which are not equipped for distributing these onto the market
[185] Electrical connectors: wiring by contacts (pins)

cannot control any on-board changes[186] besides shutting down chips or disabling communication interfaces. In our simulation model, we concentrate on the grouped system-board components $CS = \{CS_1, CS_2, \dots, CS_m\}$, which we distinguish into the five major categories *processor*, *memory*, *input/output*, *fan*, and *others*, as defined in Equation (5.16) and shown in Figure 47.

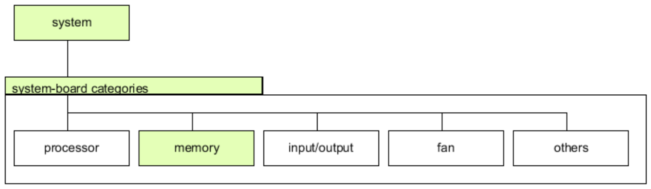


**Figure 47: System-board categories (components)**

The category $CS_1 = processor$ includes any kind of processing units; $CS_2 = memory$ refers to diverse volatile storage devices. While the category $CS_3 = input/output$ considers non-volatile storage and communication hardware, such as internal hard disk drives (HDD), solid-state disks (SSD), or InfiniBand[187], the $fan$ covers cooling devices, such as a processor fan, in-house fan, or power supply unit (PSU) fan. The category $CS_5 = others$ includes expansion cards, for instance, whereby the fraction of the *others* category on the total power consumption is negligible. We concentrate upon the *processor*, *memory*, and *fan* categories because we can manage and control these devices at the hardware level. The power consumption of the I/O devices depends substantially upon the explicit usage, e.g., either the network communication to the storage-area-network (SAN) / network-attached-storage (NAS) servers, or provides data for complex computations by an optical interconnect. We concentrate upon the small-scale and medium-scale enterprise servers. Herein, typical servers involve a small amount of I/O devices because of their less external communication in comparison with high-end servers. The I/O-based power in this class of systems is relatively small in comparison to the power consumption of the processor or memory. We therefore neglect the I/O-based power in the case of a processor-bounded or memory-bounded workload, as it has little influence on the entire power dissipation and consider them as static power. In principle, I/O-bounded workloads could be handled as well. However, in this case precise models of the respective I/O devices and their workloads would be required.

We analyze the customer-specific server system configuration $\theta_C$ automatically and consider the related category of each component. We reconfigure our configuration tree because we include the subtypes in our simulation model. Our root type $TY_0$ $(L_0)$ is the system, but the system-board categories become the level $L_1$ of our configuration tree $\theta$. We can alter the

[186] On-board changes: e.g., influence voltage or frequency, only possible at the RTL level in the design phase
[187] InfiniBand: http://www.infinibandta.org/

configuration at each level of the configuration tree. We define a rack-based server system, such as $TY_0 =$ RX200S7[188], which we want to optimize, for instance. We concentrate upon the energy-efficient hardware in our optimization strategy when we search for alternative configurations and characterizations.

**Server System Technical Specification Tree $\theta_{TS}$ – A Subtree of the Configuration Tree $\theta$**

The configuration tree concentrates upon the physical system considering the mountable system-board components. We differentiate a component itself and the respective category into the technical specification and their related characteristics $CH_{TS}$. Figure 48 shows an incomplete memory subtree that covers just the technical specification. A leaf and their ancestors[189] technically specify a component $C_i$ and define the exclusive configurations including the parent subtypes. For instance, a $PC3-12800$ memory module automatically sets the technical details, including its predecessor subtypes $DDR3-1600, DDR3$, $SDRAM\ DIMM$. The path corresponds to the explicit customer-specific memory configuration in our generic configuration tree $\theta$. The siblings[190] have alternative configurations at every certain level. We require exchangeability, such as between the $DDR3$ and $DDR4$ architecture, when we use other memory architectures in the early design phase. Thus, we do not create a new model when a technical specification changes. Instead, we extend the tree by adding a novel subtype. The encapsulation and abstraction support the required flexibility as well as scalability.

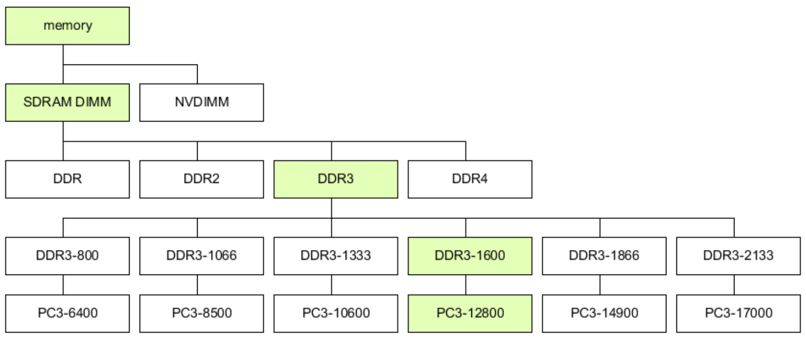


Figure 48: Technical specification tree – memory

We define the levels $L$ of our technical specification tree, as shown in Equation (5.47), and specify the root type $TY_0$ by the explicit component category $CS_i$. The subtypes $SU$ and sub-subtypes $SU_g$ define the technology characteristics $CH_{TS}$, which we consider in our optimization strategy.

[188] RX200S7: Fujitsu server system, https://sp.ts.fujitsu.com/dmsp/Publications/public/ds-py-rx200-s7-de.pdf
[189] Ancestors: vertices between a given vertex and the root vertex
[190] Siblings: vertices with same parents, also called neighbors, or adjacent vertices

$$L = \{L_0, L_1, \dots, L_k\}, k \in N_o, N_o = \{0,1,2,3,\dots\} \quad (5.47)$$

$$SU = \{SU_1, SU_2, \dots, SU_f\}, f \in N, N = \{1,2,3,\dots\} \quad (5.48)$$

$$SU_g = \{SU_{g1}, SU_{g2}, \dots, SU_{gf}\}, g \in N \quad (5.49)$$

We have grouped the characteristics of each level, which allows us to separate our optimization decision level by level. We annotate the explicit server configuration along the search path within the tree. We analyze the technical specification tree and alter the technology at each level by selecting the siblings. We specify a heuristic, wherein we begin with the level that contains the confirmed and reliable values[191] of the subtypes. Therefore, we start with the highest level[192] (leaf), explore the alternative configurations, and search the most energy-efficient ones. A lower level, nearby the root level, contains a larger amount of theoretical values in comparison to the concrete path in the tree up to the leaf. As an example, we assume the tree in Figure 48, and specify the real memory module as $DDR3 - 1600, PC3 - 12800, DDR3, SDRAM\ DIMM$ illustrated in color in the figure. We consider the set of leaves[193], which probably consumes lower energy in comparison to the actual configuration. After evaluating the siblings, we consider the parent level up to the root level. An adjustment closer to the root level becomes increasingly uncertain in comparison to a change at a higher level. Alternatively, if a server configuration limits the technology to $DDR3$ because of the system architecture, we will exclusively consider the child level in the tree instead. As a result, we neglect the upper levels in the technical specification tree and ignore the siblings of the $DDR3$ subtype. The structural restrictions of the server system lead us to start our algorithm at the leaf level that supports a higher degree of flexibility. We restrict the design space, the subtypes of each level, on the basis of the system compatibility, which reduces the alternation complexity. We accept the risk that our limited tree does not provide any ideal solution (the resulting feasible region may become empty) or we exclude a possible optimal configuration, which leads to an unsolvable optimization problem. In both scenarios, we explore the alternative configurations using a bottom-up approach. The heuristic is part of our optimization algorithm, see Section 5.4.2.1.

The technology characteristics are static simulation parameters, which result from the initial server configuration. Besides the predefined parameters, we address dynamic characteristics, which we adjust during the simulation. We exemplarily select the leaf $PC3 - 12800$ of the technical specification tree and annotate the subtypes along the path within the technical specification tree by a flag that shows the current usage in the simulation model.

[191] Confirmed and reliable values: empirical measurements, spreadsheet-based data, observations, statistical results, or customer-specific intellectual properties (IP)

[192] Highest level in tree: usually the highest level is assigned to the root and the lowest to the leaves, but we define the levels concerning the indices

[193] Set of leaves: $\{DDR3 - 800, DDR3 - 1066, DDR3 - 1333, DDR3 - 1866, DDR3 - 2133\}$

A leaf of the technical specification tree builds the root type $TY_0$ of our second configuration tree, such as the memory module $SDRAM\ DIMM, DDR3, DDR3-1600, PC3-12800$. Herein, the subtypes define the various characteristics and values of the selected component with an arbitrary order of the levels. We implicitly specify the technology characteristics of each level by selecting the leaf of the technical specification tree. We define the subtypes $SU$ in the characteristic tree in the same manner. We can replace the tree levels with each other because they build an unordered tuple of the characteristics of the explicit configuration. The colored subtypes in the figure refer to the component characteristics of the customer configuration $\theta_C$, a $registered,\ low-voltage,\ ECC,\ 4GB\ capacity$ memory module. Each tree level $L$ provides a specific characteristic, whereas the siblings refer to the possible range of values. We distinguish the static and dynamic characteristics, which we adjust in different phases in our optimization strategy.

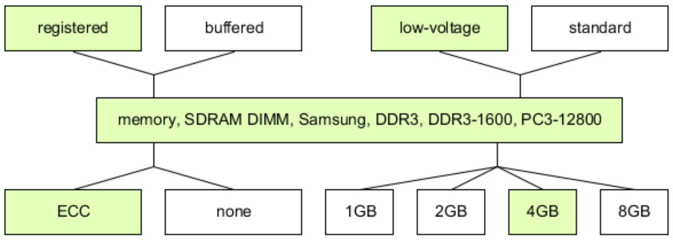


Figure 49: Configuration with characteristics and values – memory

All categories and components differ in their technical specification and their respective characteristics. We define a generic characterization tree $\theta_{CS}$ that supports the flexibility to define each component of the diverse generations. Figure 50 presents partially the generic configuration tree $\theta$, the technical specification tree $\theta_{TS}$, and the characterization tree $\theta_{CS}$. The color-marked path in the tree exemplarily represents an actual server system configuration $\theta_C$ of the memory module.

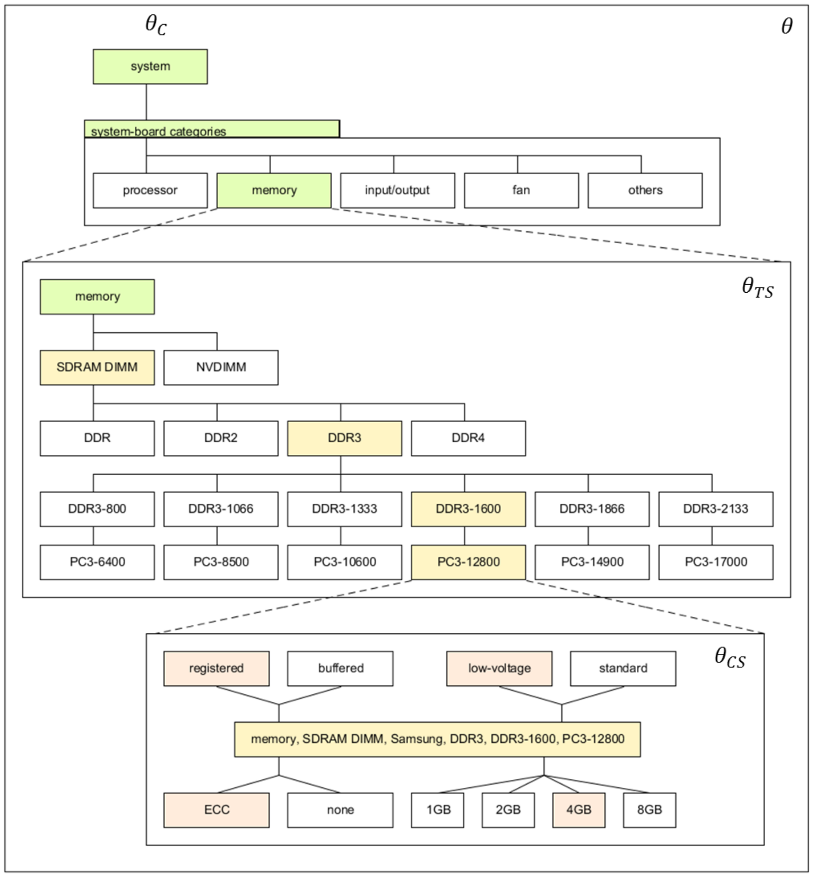


Figure 50: Technical specification and characterization tree

The design and architectural descriptions include the topology, hierarchy, component types, or connections. The architectural model $AC$ defines the configuration restrictions[194], sub-components, their hierarchy, and their electrical connectors $CC$. The system architecture and busses are fixed for a given server configuration $\theta_C$. We consider an explicit server generation and family in our simulation model. The server architecture itself restricts the components, such as the mountable motherboard, and the generation limits the component type. We differentiate the system configuration into the components, connectors, and power supply. We concentrate on the component-based concepts because we model the system architecture $AC$ and connectors $CC$ within our simulation model in Simulink[195]. We distribute the workload towards the exact number of the components. Figure 51 shows an overview of the hardware design and architecture within the configuration layer.

[194] Configuration restrictions: maximal amount of mountable components
[195] Simulink: architecture and connectors are a graphical system view designed as blocks and lines

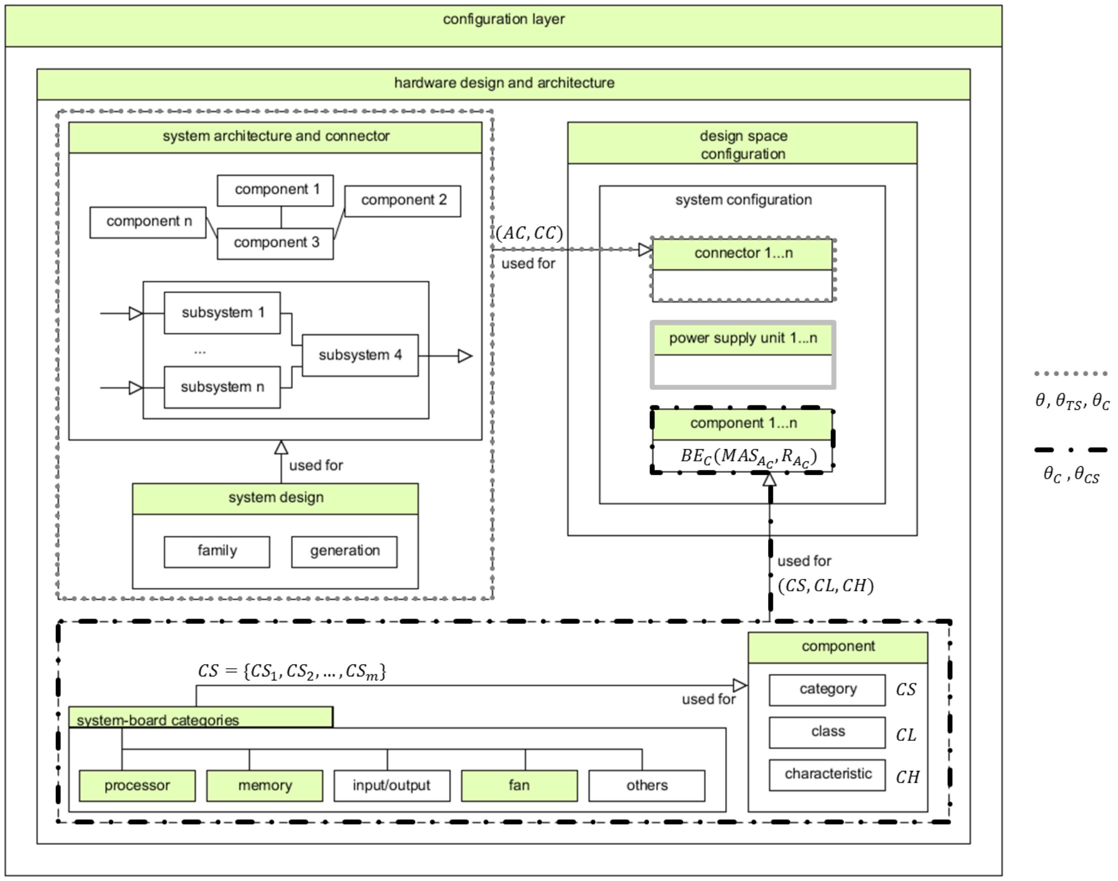


Figure 51: Configuration and characterization layer

An exception in the configuration layer is the power supply unit (PSU) of the server system, which we size adequately on the basis of the worst-case power consumption. The power supplies are more efficient in the case of higher utilization levels, see Section 3.10.3. We calculate the total power consumption $PO_{PSU_{OUT}}$ of the server system. Our PSU model considers the technical specification of the spreadsheets and the energy efficiency coefficient $EE_{PSU}$, which depends upon the supply voltage $\alpha_{volt}$ and the actual utilization specified by $PO_{PSU_{IN}}$, see Equation (5.50). In Europe, we define the supply voltage of the servers between $220V$ and $240V$[196]. The server components require the PSU power $PO_{PSU_{IN}}$ to be operational, which can be shared among the available PSUs and is specified by the PSU redundancy[197].

[196] Supply voltage: some countries operate between 100 and 127 volts, such as the USA

[197] PSU redundancy: the customer decides if all PSUs are utilized or only one PSU is utilized and the remaining PSUs are idle as long as no power failure occurs

We do not consider any redundancy settings in the case of one PSU, which is a customer-specific simulation parameter. We share the total power $PO_{PSU_{IN\_total}}$ in equal parts among the available amount of PSUs, when the PSU redundancy is set. We specify the $PO_{PSU_{IN}}$ of a single PSU and consider the remaining PSUs as idle when the PSU redundancy is not configured, see Equation (5.52).

$$PO_{PSU_{OUT}} = \begin{cases} PO_{PSU_{IN}} * EE_{PSU_{120V}}(PO_{PSU_{IN}}),\ if\ \alpha_{volt} = 120 \\ PO_{PSU_{IN}} * EE_{PSU_{220V}}(PO_{PSU_{IN}}),\ if\ \alpha_{volt} = 220 \end{cases} \tag{5.50}$$

$$PSU = \{PSU^1, PSU^2, \dots, PSU^p\}, p \in N,\ \#PSU = |PSU| = p \tag{5.51}$$

$$PO_{PSU_{IN}} = \begin{cases} PO_{PSU_{IN\_total}},\ if\ \#PSU = 1 \\ \frac{PO_{PSU_{IN\_total}}}{\#PSU}, if\ \#PSU > 1, PSU_{redundancy} \\ PO_{PSU^1_{IN}} = PO_{PSU_{IN\_total}}, otherwise \end{cases} \tag{5.52}$$

We specify the components and their behavior on the basis of the system-board categories. We characterize every component independently of the software, OS, and system architecture. Beneficially, we can use the same component definition in an embedded system or personal computer when we identify and specify the system architecture. Our model is suitable and adaptable for the blade servers, standalone servers, and embedded systems. A blade enclosure can consider more but specialized components that are connected in a cordless manner to a prewired backplane because of their smaller form factors. We adapt the static and dynamic characteristics of the components instead of the configuration because we found that not only the amount of the memory modules influences the energy efficiency, the synchronization mode is also a significant characteristic.

**Server System Characterization Tree $\theta_{CS}$**

We characterize the components of the configuration tree $\theta$ in a hierarchical system to support the aspect-based calculation methods within the matrix $MAS_{A_C}$, as defined in Equation (5.18). We define the characteristic tree, as shown in Figure 52, which is specific to each category and support the functional description in the *logical and physical* configuration layer. Herein, we support the characterization from the component down to the chip level to cover every accuracy level. Our characterization tree $\theta_{CS}$ and the respective network define the abstract models of the components within the simulation. We identify the mutual interdependencies of the system in order to create the generic models to define the component behavior, see Equation (5.32). We specify the power, thermal, and performance aspects $A_j$ of each component $C_i$ as a set of utilization-based functions $F_{A_{j_{C_i}}}(u_{cs})$ and use the hierarchical model, which provides the various categories $CS_i$ and characteristics $CH$. The categories $CS_i$, especially the components $C_i$[198], build the root type $TY_0$ of the tree $\theta_{CS}$. We specify the classes $CL$ as a technical assistance of our flexible concept, which forms the first subtype level. We further distinguish the classes in their category-specific characteristics $CH$, at the third level of the tree. We propose a dynamic structure to store the wide range of categories, classes, and characteristics, which we flexibly specify and extend in the case of the next-generation systems.

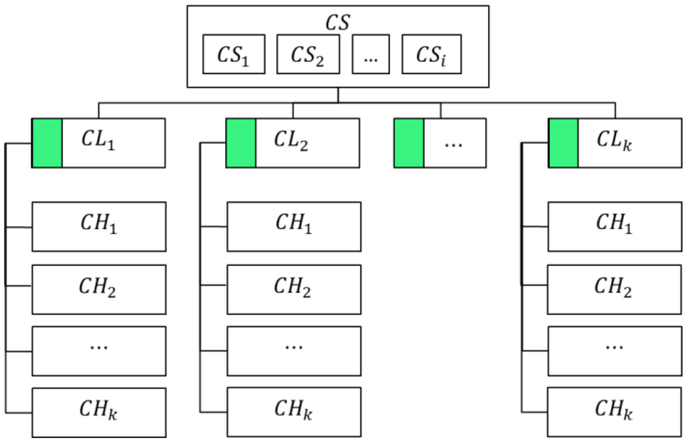


Figure 52: Category-specific classification and characterization

The calculation method of the memory power needs different characteristics than the processor power or memory temperature models. In our simulation, we require the set of all significant characteristics of each component and their relevance of the certain aspects. We consider in our characteristic tree $\theta_{CS}$ all basic classes and characteristics that are specific of any aspect. We annotate a leaf $CH_l$ by a value that will be used in the utilization-based functions $F_{A_{j_{C_i}}}(u_{CS})$ within the matrix $MAS_{A_C}$, as shown in Figure 53. We specify the characteristics $CH_l$ of the aspect $A_j$ and component $C_i$ by the weight coefficient $WF_{A_{j_{C_i}}}^{CH_l}$ to distinguish their particular significance in the calculation methods. We assign a higher weight coefficient when the characteristic has a substantial effect on the certain aspect. We concentrate upon the characteristics that are significant of more than one aspect and those whose values most affect the error rate of our calculation methods.

[198] Categories and components: We generalize the components into the categories, in this case, the category is equal to the component $C_i$.

| $CH_l$ | | |
|---|---|---|
| $val$ | | |
| $A_1$ | $A_{1_{C_1}}$ | $WF_{A_{1_{C_1}}}^{CH_l}$ |
| | $A_{1_{C_2}}$ | $WF_{A_{1_{C_2}}}^{CH_l}$ |
| | $\vdots$ | $\vdots$ |
| | $A_{1_{C_i}}$ | $WF_{A_{1_{C_i}}}^{CH_l}$ |
| $A_2$ | $A_{2_{C_1}}$ | $WF_{A_{2_{C_1}}}^{CH_l}$ |
| | $A_{2_{C_2}}$ | $WF_{A_{2_{C_2}}}^{CH_l}$ |
| | $\vdots$ | $\vdots$ |
| | $A_{2_{C_i}}$ | $WF_{A_{2_{C_i}}}^{CH_l}$ |
| $\vdots$ | $\vdots$ | $\vdots$ |

Figure 53: Annotation characteristics and weight coefficients of all components

Figure 54 shows a simplified example of the memory module, which has a *frequency* characteristic by a value of $800MHz$. We specify that the frequency is a relevant characteristic of the power, thermal, and performance models, whereby we separately weight each of them by a coefficient.

| $frequency$ | |
|---|---|
| $800MHz$ | |
| $PO$ | $WF_{PO_{mem}}^{frequency}$ |
| $TH$ | $WF_{TH_{mem}}^{frequency}$ |
| $PE$ | $WF_{PE_{mem}}^{frequency}$ |

Figure 54: Annotation aspects and weight coefficients of a particular characteristic

Equations (5.53), (5.54) and (5.55) exemplarily show that the power, thermal, and performance models of a component $C_i$ require various characteristics, which we subdivide into classes $CL$. The power consumption of a component $C_i$ is a function of the characteristics $PO_{C_i}(CH_1, CH_2, CH_3, CH_4, CH_7, CH_8, CH_9)$ concerning the related classes $CL_1, CL_2$, and $CL_3$. We differentiate on the class-specific characteristics in our characteristic tree because some refer to the configuration or the technical specification, which are static settings of our simulation. We define the specific characteristic and classes to specify the accurate models in Section 5.2.2.

$$PO_{C_i} = \begin{cases} CL_1 \rightarrow CH_7, CH_8, CH_9 \\ CL_2 \rightarrow CH_1, CH_2 \\ CL_3 \rightarrow CH_3, CH_4 \end{cases} \tag{5.53}$$

$$TH_{C_i} = \begin{cases} CL_1 \rightarrow CH_2, CH_3 \\ CL_2 \rightarrow CH_4 \end{cases} \tag{5.54}$$

$$PE_{C_i} = \begin{cases} CL_1 \rightarrow CH_2, CH_3 \\ CL_3 \rightarrow CH_3, CH_4 \end{cases} \tag{5.55}$$

In the next section, we analyze the static $ST_s$ and dynamic $DY_s$ characteristics of the server system $SY$. Our aim is to characterize the system and the corresponding components within our configuration tree to support the calculation methods of each aspect, as shown in Equation (5.56). We conclude the configuration and characterization in $\beta$ for better readability.

$$[EX] \quad \begin{array}{c} \begin{bmatrix} ST_s \\ MAS_C, R_{A_C}, R_{BE} \end{bmatrix} \\ \uparrow \\ \overbrace{\begin{bmatrix} \theta_C \\ CS_i, C_i \\ CL_l \\ CH_l \end{bmatrix}} \\ \uparrow \\ \beta \end{array} \rightarrow [PO\ PE\ TH] \tag{5.56}$$

We specify declaratively the heterogeneous aspects of all components using encapsulated layers to create a generic system model that is suitable for diverse academic and industrial full-system approaches. We abstract from irrelevant features because of their little effect on energy efficiency, or the features that are specific to a single server system. We characterize the server system $SY$ in the physical domain, which includes the system architecture, hardware, configuration, and relevant characteristics. We include academic results in our aspect-based models. According to [RRK 2008], we cover processor-dominant systems, such as file servers, and alternative configurations for standard server types. Furthermore, we support the existing measurements to get a precise model of an explicit server configuration, including the certain characteristics. We strictly separate the modeling and simulation characteristics.

### 5.2.2 Configuration – Characterization of the *Logical and Physical* Layer $SY(\theta)$

The *physical and logical* layer provides all significant data that a power, temperature, or performance model requires. Fully attributed data is fundamental to support the related calculation methods. In this section, we define the detailed models of the components within the configuration tree, which we characterize to calculate the respective values. We analyze and clearly identify the relevant characteristics at various domains that support the flexibility to use states, transitions, or instructions. Our concept considers diverse academic approaches to provide a sufficient accuracy level for each component. The component description in the *logical* domain can represent tasks, routines, functions, programs, procedures, activities, operations, or interactions. In contrast, the *physical* domain describes functional blocks of a component without any interior definition. The *logical and physical* configuration layer defines electrical (power, energy, performance), mechanical (motor, rotation), or thermal (heat transfer) behavior. Herein, we concentrate on the flexibility and the scalability to characterize every aspect at each domain separately. We define the power models to be independent of the temperature or performance model that refers to an actual state. We cover all component states that form the base of our calculation methods. In contrast to [RL 2007], we define our component models on the basis of the utilization levels.

In principle, we use the modeling flow of [FCM 2014] because we specify the component-specific power consumption and the system behavior and we combine them with the hardware design to simulate the entire system. We specify the states of each component $C_i$ at the *logical and physical* configuration layer. We support various server system domains, such as a chip or component level, because we use cycle-based as well as instruction-based models to address the functional level. The authors of [Che 2006] create a component model on the basis of instructions and accesses, which we abstract on an algorithm or architectural level because we consider the generic workloads and avoid the instruction-based details. The authors in [Dre 2006] present a Gajski Y-diagram, which illustrates the logical, component, and system domain. For each domain the authors define the behavior, geometry, and structure. In the same manner, we specify differential equations of the physical domain, algorithms for the block definition of a subsystem, or technical specifications for our components. Secondly, we define the domain-specific and aspect-related models using a mix of spreadsheets, measurement results, and vendor-specific data. We map the component behavior models towards the entire system configuration.

In addition, we abstract a component when just spreadsheets or empirical measurements exist, usually for next-generation systems. If a vendor provides only spreadsheet-based data, an over-estimation is better than an intuitive prediction, or any values [Fuj 2012]. The authors of [BHS 1998] propose an overall black-box concept, which abstracts all internal resources, but overestimate power. We do not restrict ourselves to using a black-box model only, because of the insufficient accuracy. Beneficially, the black-box concept enables the power estimation for novel components regardless of missing architectural or structural details. The authors of [Hag 2009] propose a flexible description of the component models. In addition to the description, we propose an entire technology mix that combines the benefits of the black-box, gray-box, and white-box models to provide an adequate accuracy level. If we know the inner component structure and instructions, we create a precise white-box model. In contrast, we emulate a black-box component, as a functional block, when the internal behavior is unknown. [GFN et al. 2006] state "we have to,…, decide for each (sub) component whether it should be regarded as a 'black box' or as a complex component." The authors decide according to the significant contribution to the total power consumption and the corresponding component complexity. Our approach is potentially more precise in comparison to [Hag 2009]. Furthermore, we reuse the mixture of the algorithmic and architectural parameters, as shown in [JLS et al. 2003, LJS et al. 2004], whereas the algorithmic parameters rely upon the software execution but the architectural parameters are independent of the software. We consider the architecture to be within the configuration layer. Figure 55 shows our behavioral component models that include the aspects-based models, such as performance, power, and thermal. We create each model regarding the available inputs $IN$ and the certain parameters $PA$ of each component. The output $OUT$ of each model can either be a function of the inputs and parameters, a function of the corresponding component states[199], or an internally specified function $F$, as shown in Equation (5.61). A set of scalable inputs and parameters help to create a suitable aspect-related method, either with instructions on the functional-level, or abstracted behavior as a black-box model.

$$IN = \{IN_1, IN_2, \dots, IN_l\} \tag{5.57}$$

$$PA = \{PA_1, PA_2, \dots, PA_l\} \tag{5.58}$$

$$s = \{s_1, s_2, \dots, s_k\} \tag{5.59}$$

$$F = \{F_1, F_2, \dots, F_k\} \tag{5.60}$$

$$OUT = \begin{cases} f(IN, PA) \\ f(s, LUT) \\ f(IN, F) \end{cases} \tag{5.61}$$

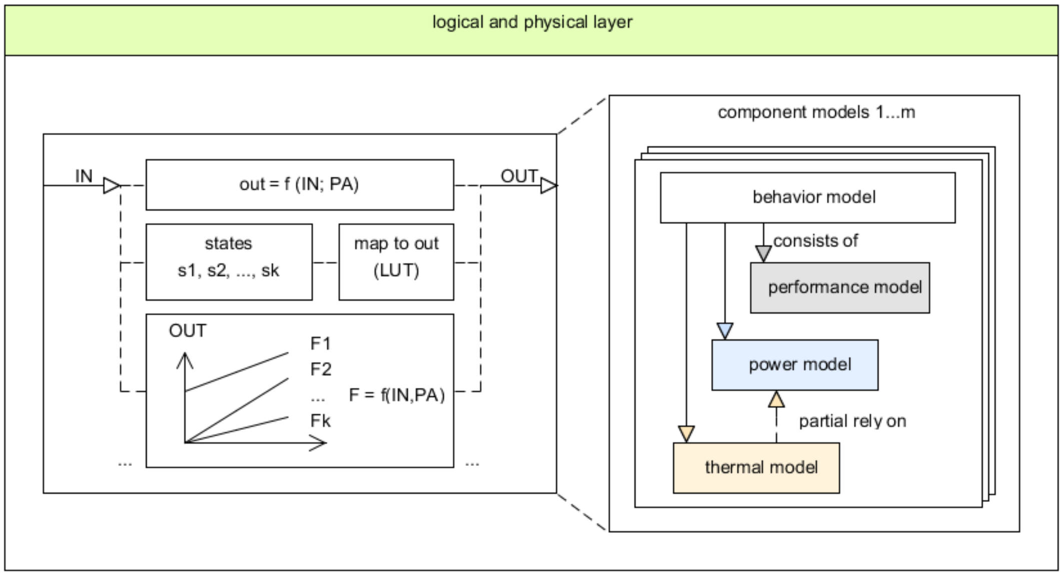


Figure 55: Logical and physical configuration layer, adapted from the original in [Hag 2009]

[199] States: define the value in a lookup table (LUT)

For a *logical and physical* definition of each component, we eliminate insignificant parameters and reduce the output uncertainty. We concentrate upon the input, which generates and correlates most of the output variables. In contrast, to the parameter sensitivity analysis [Ham 1994, LB 2005], we define the component model and its dependent parameters without considering a probability density function to each input parameter. We analyze diverse spreadsheets, measure components with changed settings, and use benchmarks to find the relevant parameters, and their consequences to the output. We assume multi-parameter impact when we cannot clearly identify the relation of a single parameter to a certain aspect.

We analyze a component on the basis of a software trace and we monitor the synthetic benchmark, which correlates to an explicit component state. We create a linear regression model, as done in [BC 2010, MAC et al. 2011], considering the parameters and analyze the accuracy. Another approach in [Riv 2008, RRK 2008] analyzes the activities on a cycle-by-cycle basis executing a particular workload. Both approaches are improper when we consider next-generation systems.

We define an input matrix of the significant parameters of each component and separately configure their relevance. We extrapolate results for next-generation server systems on the basis of our findings of current servers used today. We consider technology trends such as shrinking the die[200] type (size) or increasing the power density[201]. As a consequence of the exponential power increase at each frequency step, the vendor shrinks the logic on a single die. The performance increases, but implies growing temperatures, and a higher cooling effort in comparison to predecessor generations.

We present the characteristics to calculate the diverse component aspects and define the component behavior dependent on the technical specification. We exemplarily present the findings of our processor and memory analysis. In general, we study all characteristic combinations of each component separately to identify their relevance, which we define as weight coefficients. We specify the relevant characteristic of our configuration tree.

We determine the category-based and aspect-based characterization of all stages[202] within the product life cycle (PLC), which is a challenge for our generic approach. In the early design phase, we have less data of the components, which usually are available by means of a paper base or a prototype. The vendor cannot ensure compliance unless the production ramp-up starts with a pilot release. In this case, we gain data by extrapolation and interpolation of the spreadsheets and try to compensate the data inaccuracy through measurement results based

[200] Die: dice, physical chip separated from a wafer, small area (array) of semiconductor material, creates the integrated circuit
[201] Power density: growing amount of transistors in a chip
[202] PLC stages: market introduction, growth, maturity, saturation, and decline

upon previous generations. The amount of the component data[203] and their accuracy increases over the entire life cycle, which is separately evaluated and validated. Our major challenge is to sufficiently generate models on the basis of the available data on various system domains for the different aspects, see Section 3.3.

**Category-based and Aspect-based Characterization**

We present the characteristics to calculate the diverse component aspects $A_j$. We define the component behavior dependent on the category, their structure, or technical specification. A category-based characterization aims at supporting an independent and detailed description of the component behavior. We can reuse explicit benchmark results, which we observe from the academic approaches for a specific component under certain conditions. We consider studies of isolated components and develop a generic model that covers the individual characteristics. We combine the heterogeneous benchmark results to examine various hardware configurations and finally the entire system behavior. In this section, we exemplarily present our observations of a few memory modules and processors that result from our adjusted version of the parameter sensitivity analysis. We define the component characteristics and their relevance for each aspect $A_j$.

We empirically analyze existing spreadsheet-based data and measurements, which are helpful to determine the characteristics and find their weight coefficients, see Appendix A3d and A3f. We analyze single characteristics, such as the vendor, which can be a cause for variations in the power consumption of an equally described component. We define the weight coefficients, especially for these findings. We cannot examine all characteristics separately, because a couple of characteristics depend upon each other. In this case, we define a system of equations to consider their dependencies.

**Memory Power Characterization**

At first, we describe the relevant characteristics of the memory modules to develop an accurate power model. Herein, we analyze the memory spreadsheets to define the significant characteristics. The memory vendor *Micron* [Mic 2007] distinguishes the entire power of a $DDR3$-module into the total background power $PO_{back}$, active power $PO_{act}$, and operating power $PO_{op}$, see Equation (5.62).

$$PO_{mem} = PO_{back} + PO_{act} + PO_{op} \tag{5.62}$$

[203] Component data: spreadsheets (technical specification) or measurements

The background power is the lowest power state and always occurs[204] during normal operations. The vendor specifies the background power by the percentage of the time all banks are precharged, the percentage of the bank precharge time, and the percentage of the active time. The active power includes the time to select a bank or a row address for storing the data in the memory array. The corresponding command requires a number of clock cycles to activate a cell within the array that can be written or read. The operating power defines the power for each operation, which the vendor distinguishes into the read, write, data line[205], and termination operation. We cannot specify the memory power when we do not have any knowledge about the explicit memory operations (read-to-write ratio) or the entire workload. The clock cycles and timings depend upon the memory generation. We cannot predict these comprehensive details of next-generation memory modules.

In the second step, we check the commercial tools for the memory variations and their customer-specific characteristics, which we analyze in our third phase that includes the vendor-specific measurements. We present the results for a group of registered[206] memory modules provided by a certain vendor $A$. The registered or unbuffered characteristics are known as synchronization modes, which influence the controller access and memory performance. The registered modules include an extra register between the memory controller and the chips on the module. An unbuffered[207] memory module enables the direct access on each chip individually and in parallel, which always consumes less power than a registered module with the same technical specification, see Appendix A3d. Our internal results show that the power consumption of two modules with an equivalent technical specification[208] differs up to ten percent. We also concentrate on the vendor $A$-based memory modules[209], which in general consume less power in comparison to vendor $B$ memory modules, as shown in Appendix A3d and analyzed in [RLG et al. 2008] of the $DDR2-SDRAM$ generation. We analyze the memory frequencies of a wide range of modules. The frequency[210] changes from $f_k = 533MHz$ to $f_{k+1} = 667MHz$, which has the largest impact with approximately 20 percent of the power consumption. Frequencies lower than $f_k$ or higher than $f_{k+1}$ increase the power consumption less than ten percent when the frequency switches to a higher value.

[204] Background power: typically occurs when all banks are precharged
[205] Data line: DQ lines, data width, input/output pins (rank linking)
[206] Registered (buffered) module: synchronize the timings between the address and control lines, see Appendix A3d
[207] Unbuffered (regular or unregistered) module: has diverse input lines on the same module with various loadings
[208] Equivalent technical specification: memory module DDR3-SDRAM 1GB 1R A, vendor $B$, (registered vs. unbuffered)
[209] Memory vendors: common server memory module vendors are *Micron*, *Hynix*, *Netlist*, *Qimonda*, and *Samsung*
[210] Memory frequency [MHz]: in the interval $[400,533,667,800,933,1066]$, also defined as equivalent transfer rate (speed-bin, throughput) in the interval $[800,1066,1333,1600,1866,2133]$

The related die types are relevant characteristics that specify the component revision (technology) denoted by a letter in the interval $[A, B, C, D, E, F, G, H, J, L, M, N]$. A die type $d_k$ that is closer to the interval's beginning, consumes more power than a die type $d_{k+l}$ at the end of the interval. An actual component revision of the memory module is usually more energy-efficient in comparison to an older date of manufacture. The fabrication size[211] results from the die type, which we cannot consider separately. Furthermore, the die type is the significant characteristic when we have both the capacity and rank[212] linking[213] (e.g., $x4$, $x8$, $x16$) technology. Herein, we found a decrease of approximately 50 up to 60 percent, whereby we neglect the frequency. When we double the memory density $[GB]$ and ranks $[R]$ at the same time, the power increases approximately ten percent from $1GB, 1R$ to $2GB, 2R$ and nearly 20 percent from $2GB, 2R$ to $4GB, 4R$. We examine that a doubled density, but a halved rank linking, result in a power increase of approximately 70 percent. In this case, the frequency influences the power consumption by nearly two percent. We conclude that the memory characteristics (the capacity, rank, rank linking, and density) rely upon each other. Therefore, we define them as a system of linear equations, which we describe in the implementation section.

The findings from the spreadsheets and measurements enable a better understanding of the characteristics to develop an adequate memory model. We distinguish into the technical specification $CH_{TS}$ and memory characteristics $CH_{CFG}$, as graphically shown in Figure 48 and Figure 49. The technical characteristics are a result of our analysis on the commercial tools and our decision to cover a wide range of memory technologies. The customer pre-defines the memory modules on their generation and capacity. Additionally, we divide the characteristics $CH_{CFG}$ into the static $CH_{CFG}^{ST}$ and dynamic $CH_{CFG}^{DY}$ configuration, as shown in Equations (5.63) and (5.64).

$$CH = \{CH_{TS}, CH_{CFG}\} \tag{5.63}$$

$$CH_{CFG} = \{CH_{CFG}^{ST}, CH_{CFG}^{DY}\} \tag{5.64}$$

We analyze the characteristics and define their relevance using the weight coefficient $WF$. Thus, we develop the memory power model with the weight coefficient $WF_{TS}$ of the technical specification and the weight coefficient $WF_{CFG}$ of the configuration. We distinguish in the static and dynamic power of all components within our system, as addressed in [SIC 2003, GFN et al. 2006]. As a result, we introduce a static and dynamic weight coefficient of the configuration characteristic, see Equations (5.66) and (5.68). The authors of [GFN et al. 2006] define various correlation functions $F_{SC}$ between each component $C_1, \ldots, C_i$ and their power functions $F_{1\ldots i}$ . of each component by a set of dynamic and static parameters.

[211] Fabrication size: nanometer technology [nm]
[212] Ranks (banks): a group of chips at the memory module having the same chip select
[213] Rank linking: number of chip's output pins (bit wide, data width), see Appendix A3d

In contrast to [GFN et al. 2006], we do not concentrate on the component-based power functions. Our method includes thermal and performance definitions to cover the real behavior.

$$PO_{mem} = PO_{TS} + PO_{CFG} \tag{5.65}$$

$$PO_{mem} = CH_{TS} * WF_{TS} + CH_{CFG} * WF_{CFG} \tag{5.66}$$

$$PO_{mem} = PO_{TS} + (PO_{CFG}^{ST} + PO_{CFG}^{DY}) \tag{5.67}$$

$$PO_{mem} = CH_{TS} * WF_{TS} + (CH_{CFG}^{ST} * WF_{CFG}^{ST} + CH_{CFG}^{DY} * WF_{CFG}^{DY}) \tag{5.68}$$

According to [TMW 1994, SIC 2003, AR 2016], the static power forms the basic power of an inactive component. The authors of [Bel 2001] address the leakage power as a static characteristic, whereby the leakage power relies upon the capacitor's size.

$$PO_{CFG}^{ST} = P_{leakage} = V_{DD} * I_{leak} \tag{5.69}$$

In the next sections, we briefly describe academic memory power models[214] or their corresponding characteristics. We specify the dynamic power by the states in various abstraction levels, as shown in [BM 1995, BHS 1998, YVK et al. 2000, BJ 2003, TRJ 2005, Han 2007, HJZ et al. 2008], but neglect the transition $t_{kl}$, which switches a state $s_k$ to another state $s_l$, because the transition time is much smaller than the time within a state. The consumed power within a microsecond or millisecond is negligible in comparison to the power within a state on second base. We assume the transition power as offsets.

$$PO_{CFG}^{DY} = P_{states} + P_{offset_{transitions}} \tag{5.70}$$

The authors in [IM 2003, LEU et al. 2010, HCE et al. 2011] address the memory access-rates either for read or write accesses to estimate the memory power. We do not exactly know the read and write accesses of every workload. We abstract the accesses into the utilization level and map them to a probability of read, and write accesses. We correlate the various states to the switching frequencies or instructions using the utilization levels and read-to-write ratio as a software-based setting. We group and abstract the explicit memory accesses to reduce complexity. Furthermore, we define a read-to-write ratio to compensate concrete accesses, whereas we differentiate into full write, read, or mixed operations to cover various memory workloads. We toggle between the operations, which will be more precise in the case of memory-intensive utilization levels. We define the dynamic characteristics, as shown in Equation (5.71).

$$CH_{CFG}^{DY} = \{voltage, frequency, utilization\ level, read - to - write\ ratio\} \tag{5.71}$$

[214] Academic memory power models: see Section 3.4

The work in [GKG 2012] additionally considers diverse memory patterns, which we neglect. A memory operation requires data transfer from the addressed memory cells[215] to the bus, which builds the dynamic power consumption. Herein, Bellosa [Bel 2001] considers several low-power memory states. We extend our memory power model considering the low-power states. The memory spreadsheets define the current states, denoted by $IDD$[216] and a corresponding number. Similar to the die type, the current $[mA]$ depends upon the state's location in the interval $[IDD0, IDD1, IDD2P, IDD2N, IDD3P, IDD3N, IDD4R, IDD4W, IDD5B, IDD6, IDD7]$. The numbers differ by the memory vendor and generation. An $IDD$ closer to the interval's beginning (e.g., $IDD1$) consumes less power than an $IDD$ at the end of the interval, such as $IDD6$. The power difference between the various memory states is negligible and requires a complex model, which increases the calculation time. We summarize that only a few memory states are significant while executing the benchmarks or working in the real world. Therefore, we group the major $IDD$ states to be considered into the idle state and the active state, which we further distinguish in the refresh mode or read-to-write mode. Another memory characteristic is the interleaving method for the dual in-line memory modules (DIMMs). The interleaving characteristic defines the symmetric memory usage, whereby the data moves between the various memories addresses. The channel interleaves divide the memory blocks and spread the data across all channels. The bank interleaves define the parallel usage of the memories. The rank interleaves enable the accesses of a memory rank while another is being refreshed and provide the request parallelism, which results in a better performance. The non-uniform memory access (NUMA) is an asymmetric memory configuration, which is configurable when we disable the interleaving. The work in [BS 1976, ZZX 2000] shows that the interleaving method influences the system's performance. [Tol 2009] argues, "Interleaved memory systems map contiguous cache lines to multiple devices, breaking the one to one relationship between devices and physical address space used in our earlier implementation. Consider the earlier example memory system composed of eight devices each with 1Gigabyte capacity under interleaving. Even though the physical devices have the same capacity, the mapping of devices into the physical address space is different. Whereas a 1Gigabyte device is mapped to a specific 1Gbyte region within the physical address space in a sequentially mapped system, two or more different memory devices may be mapped to the same 1Gbyte region in an interleaved system". The average execution time varies because of the cache line interleaving, the page interleaving, or other methods. We assume the interleaving methods as dynamic characteristics.

The authors of [KCB et al. 2013] state that capacitance, frequency, and data width are significant characteristics for power consumption of the dynamic random-access memory (DRAM). Thus, we include the capacitance and data width as a static configuration. The work in

[215] Memory cells: capacitors that store data
[216] IDD: drain current of a CMOS circuit, notation: a letter after the number specifies a sub-state

[Bel 2001] presents the memory size (capacity) as a static parameter for the power consumption. The authors of [ZX 2012] present that the leakage power of DRAMs exponentially increases by the memory capacity. The most relevant characteristic in [KGS 2008] is the total system memory capacity. The authors present a linear relation between the increasing memory size, such as the amount of DIMMs, and their related energy efficiency. We consider the amount of memory modules and their respective capacity in our configuration tree. We reuse the findings in [KGS 2008] to define the weight coefficients of the memory capacity. In conclusion, in our memory power model we integrate the academic characteristics, summarized in Table 26 and Table 27, which are based upon spreadsheets and measurements.

**Table 26: Memory characteristics $(I)$**

| Memory characteristics | [Bel 2001] | [HCE et al. 2011] | [KCB et al. 2013] | [LEU et al. 2010] | [IM 2003] | [ZX 2012] | Our concept |
|---|---|---|---|---|---|---|---|
| **Static characteristics** | | | | | | | |
| Capacitors / capacitance capacity (size) | y | | y | | | y | y |
| Quantity (#) | | | | | | | y |
| Vendor | | | | | | | y |
| Generation | | | | | | | y |
| Family | | | | | | | y |
| Series | | | | | | | y |
| Density | | | | | | | y |
| Die (component revision) | | | | | | | y |
| Fabrication size (nm) | | | | | | | y |
| Synchronization mode | | | | | | | y |
| Module ranks, rank linking (data width) | | | y | | | | y |
| Timings | | | | | | | y |
| Resistance | | | | | | | y |
| Interleaving | | | | | | | y |
| Refresh | | | | | | | y |

Considered (y), not considered or unknown (no entry)

Table 27: Memory characteristics $(II)$

| Memory characteristics | [Bel 2001] | [HCE et al. 2011] | [KCB et al. 2013] | [LEU et al. 2010] | [IM 2003] | [ZX 2012] | Our concept |
|---|---|---|---|---|---|---|---|
| Dynamic characteristics | | | | | | | |
| Frequency | | | y | | | | y |
| Voltage | | | | | | | y |
| Accesses / instructions / operands | | y | | y | y | | |
| Error correction | | | | | | | y |

Considered (y), not considered or unknown (no entry)

**Processor Power Characterization**

We analyze the relevant processor characteristics. In the early design phase, the processor vendor specifies a novel family and series with respect to the thermal design power (TDP) within their spreadsheets. The vendor restricts the details of the processor that is accessible to the public. Our challenge is to estimate the processor power consumption on the basis of the rudimentary technical specification, especially for next-generation systems. We plan the entire system and the size of the power supply unit (PSU) in the first product life cycle stage. One problem is that we cannot specify a power curve when we only have the largest power consumption on the basis of the thermal design power. We need further data regarding the processor power consumption, e.g., under ideal conditions for every intermediate utilization level. Usually, the vendor roughly defines the power consumption by idle, average, and full utilization levels for business and collaborative partners. For our study, we did receive the internal spreadsheets of our processor family considering the power consumption at certain utilization levels, based upon a cooperation with Fujitsu Technology Solutions GmbH[217].

We analyze the spreadsheets of the *E5-2600* product family that specifies the maximal frequency $f_{max}$[218]; but the technical specification does not include the amount of p-states[219], which helps to identify the idle or average frequencies. The *E5-2600* product family always has a minimal frequency of $f_{min} = 1.20GHz$, but we do not have any knowledge of the corresponding p-state to estimate the power on the basis of these data. We found that the frequencies change in equidistant steps of $\Delta f = 0.1GHz$, which is specific to the x86-

[217] Fujitsu Technology Solutions GmbH: http://www.fujitsu.com/fts/

[218] Maximal frequency: at the lowest p-state, in general $P_1$, but with turbo mode $P_0$

[219] Amount of p-states: unknown for novel components in the early design phase, otherwise readable by the OS or special processor tools which support ACPI

architecture. Thus, we calculate the amount of p-states $k$ to define the manageable frequencies for the power curve using the quotient of the frequency range $f_{max} - f_{min}$ and the step size $\Delta f$, as shown in Equation (5.72). In Appendix A3f, the processor $C_1$ has a maximal frequency of $f_{max} = 3.0GHz$ and consequently 18 p-states, for instance. We formally define the possible frequency interval in Equation (5.73) with $k \in N_0$. The number of p-states defines the power resolution of each utilization level.

$$k = \frac{f_{max} - f_{min}}{\Delta f} \tag{5.72}$$

$$f = [f_{max}, f_{max} - 1 * \Delta f, f_{max} - 2 * \Delta f, \dots, f_{max} - (k-1) * \Delta f, f_{min}] \tag{5.73}$$

We require the manageable frequency interval and the adequate amount of p-states in our concept to estimate the power range, especially when we vary the processors and their characteristics. The definition of the frequency interval improves our matching process between the utilization levels and the related frequencies when we map the workload to the processor. We distinguish in our concept whether we fully utilize one processor or distribute the workload across multiple processors. Our approach runs into problems when two processors have the same maximal frequency but different p-states, and vice versa. A reason is that the size of the equidistant steps $\Delta f$ depends upon the processor generation, family, or series. Another challenge is that the voltage-frequency pair is not unique. Two fully utilized processors may have different voltage-frequency pairs, as shown in Equation (5.74). Each processor has a flexible amount of p-states, and the architecture limits the maximal frequency $f_{max}$. Our model covers varying p-states instead of directly calculating the power, assuming the concrete voltages or frequencies. We consider the relations between the processor frequencies and their particular p-states at a certain utilization level to be independent of the explicit voltage-frequency pairs.

$$CPU_k(100\%) \rightarrow (2.0\text{V}, 2.5\text{GHz}) \qquad CPU_{k+1}(100\%) \rightarrow (1.2\text{V}, 2.0\text{GHz}) \tag{5.74}$$

We specify the processor power on the basis of the rudimentary data, such as the frequency interval, under the condition that the vendor does not define the minimal power. The thermal design power (TDP) indicates the upper power limit and corresponds to the maximal frequency $f_{max}$. Our next challenge is to estimate and interpolate the dynamic processor power upon the basis of the frequencies or the utilization levels, respectively. In the early design stage, we collect and aggregate the data considering previous studies to transfer the gained experiences into our model. Alternatively, we measure the power of an available predecessor generation or a processor who is almost identical and transfer the results to our present processor. In both cases, we analyze the power consumption to specify the slope $\Delta PO_{proc}$ of the power curve, which linearly defines the power ascent or descent, see Appendix A3f. We assume that the processor power curve of the most recent generation, which we select as a basis, behaves approximately proportional to our next-generation processor. We define the power correction

$PC_{P_k}$ of the present processor as a product of the slope $\Delta PO_{proc}$ and the related p-state $k$, see Equation (5.75). The processor power consumption is a nearly linear function, which we specify as a manageable power interval by the subtraction of the power correction $PC_{P_k}$ of each p-state beginning from the maximal thermal design power and ending with the minimal power at the highest p-state[220] , as shown in Equation (5.76).

$$PC_{P_k} = k * \Delta PO_{proc} \tag{5.75}$$

$$PO_{proc} = [TDP, TDP - PC_{P_2}, TDP - PC_{P_3}, \dots, TDP - PC_{P_k}] \tag{5.76}$$

We define the dynamic fraction within the power interval through the power corrections of all p-states $[PC_{P_2}, PC_{P_k}]$. Finally, we can estimate the processor power consumption on the basis of rudimentary vendor-based data (spreadsheet) by weighting the power of a similar processor, or by extrapolating the values of the predecessor generation in the early design stage for the next-generation processors.

In a second step, we prove our estimation-based approach using the assigned spreadsheet data of the processors, which are already introduced into the market[221]. We check the accuracy, especially the applicability of our approach, and exemplarily analyze a subset of the *Intel Xeon* processors[222]. We estimate the power consumption of the processors, whereas we assume that the processors do not exist. We compare the updated technical specification and check which characteristics have been adjusted. We analyze the power consumptions and the impacts of the characteristic variations in both life cycle stages. We found that our linear regression approach overestimates the slope of the power curve by nearly 25 percent in relative upward deviation in comparison to the firmly defined values in the vendor's spreadsheets of the released processors. Thus, we overestimate the entire power consumption of the processors in the early design phase. If we neglect the outliers[223], our upward deviation decreases to approximately nine percent. As all these outliers have more than 17 p-states, our method seems to be inaccurate for processors with more than 17 p-states. We adjust our calculation methods concerning the non-linear power consumption. At the same time, we analyze our under-estimation of the power, which is negligible because of a relative deviation of nearly three percent. An over-estimation is more critical in comparison to the slight under-estimation when we size the power supply, which results in an extra power overhead, or the inefficiency of the PSU, as described in Section 4.4.

[220] Highest p-state: lowest frequency
[221] Market introduction: first product life cycle phase, release of a server system
[222] *Intel Xeon* processors: *E5-2600* product family ($C_1 - C_{17}$), see Appendix A3f
[223] Outliers: processors ($C_1$, $C_{13}$, $C_{14}$, $C_{15}$), a subset of the observed processors ($C_1 - C_{17}$)

In a third step, we validate our approach concerning the commercial server system calculators, see Section 3.10.3, and concentrate upon the processor power calculation to check the consistency. We study the entire power of the systems because the industrial tools do not provide the processor power as a separate value. We analyze the results by varying the processor characteristics or hardware configuration, such as the generation, family, and series. Our estimation method for a wide range of processor families is almost identical with the industrial tools, which rely upon the spreadsheets. We exemplarily check the *E5-2600* processor family, wherein we overestimate the processor power of approximately seven percent in relative upward deviation. We assume that the commercial calculators consider a similar linear regression method to estimate the processor power consumption.

In a next step, we observe the measurements of the existing processors in an actual environment to check whether the spreadsheet data correspond to the reality and to keep the error rate of the over-estimation as low as possible. We determine a decreasing power gap between our process of estimating and measuring. Figure 56 and Figure 57 exemplarily show the estimated and measured power curves of the processors *E5-2690v2* and *E5-2670v2*. The measured power is approximately 30 percent of the spreadsheet-based power at the idle utilization level. At a utilization level of 50 percent, the measured power is nearly 60 percent of our estimated value. Finally, the power gap is less than ten percent at the full utilization level. Our challenge is to improve the estimation process, especially at lower utilization levels, so that we can define the power consumption in the early design phase more precisely. The lower utilization of the processor is significant for the memory-bounded workloads.

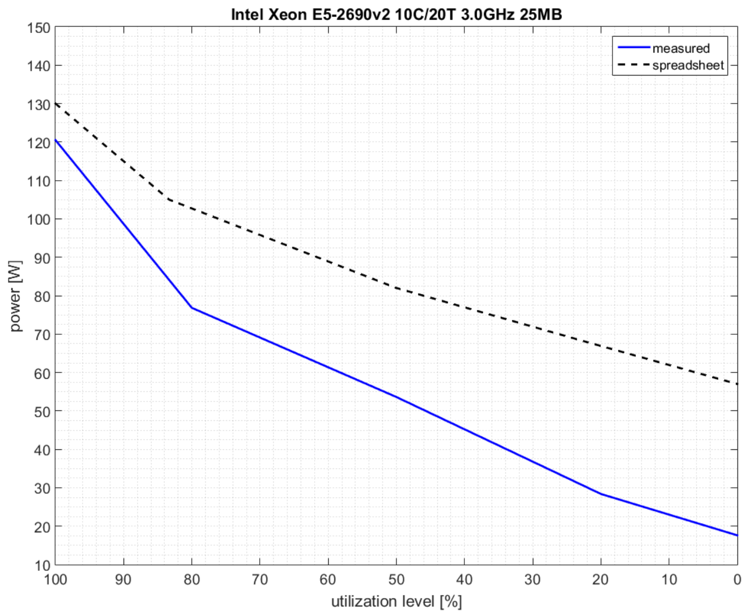


Figure 56: Spreadsheet-based estimation vs. measurements of the processor $(C1)$

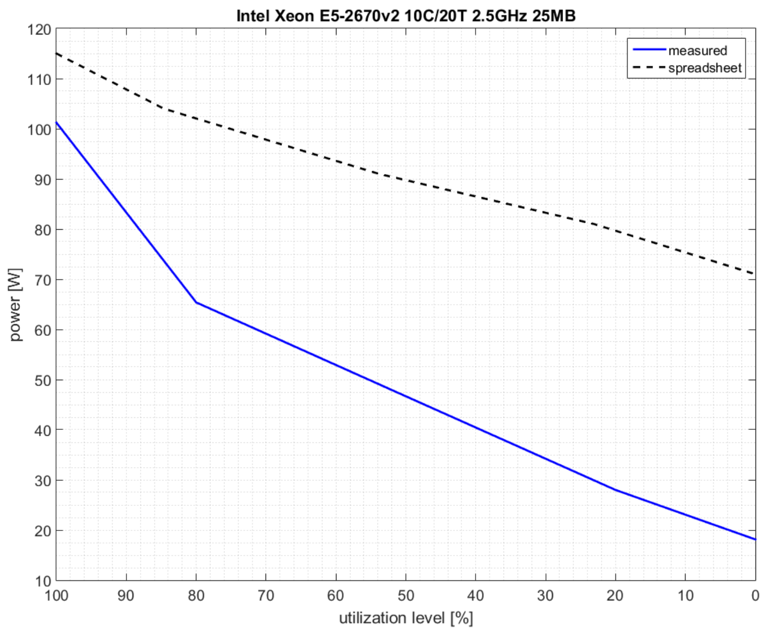


**Figure 57: Spreadsheet-based estimation vs. measurements of the processor $(C3)$**

We found that the vendor, the commercial tools, or our spreadsheet-based estimation method always overestimates the processor power consumption, as shown for the product family *E5-2600* in Appendix A3f. We optimize the calculation method to close the gap between the spreadsheet and measurement values because we conceive a non-linear processor power model considering the technical specification $CH_{TS}$ and the dynamic characteristics $CH_{CFG}^{DY}$ to be aware of the various utilization levels. We specify the static and dynamic characteristics and refine our spreadsheet-based approach by including our findings of the non-linear behavior of lower utilization levels.

In the next phase of our analysis, we determine the characteristics of diverse processor generations that are available at all product life cycle stages. We compare the similar processor generations and their power consumptions, which helps to identify the critical characteristics. We restrict ourselves to the specific x86-based *Intel* processor characterization because today's server processors are more energy-efficient and performance-optimized in comparison to the AMD processors, which are uncommon in present server systems. We exemplarily describe the relevant characteristics of the third *Intel Xeon* generation and architecture, code name *Ivy Bridge*, which involves the *E5-2600v2* processor family with a fabrication size of 22nm. The quick path interconnect (QPI), a static characteristic, is a point-to-point interconnect between one (or more) processors and the memory controller. The processor power consumption increases by the QPI transfer rate, especially at the idle utilization level, see Appendix A3f. We analyze the processors with identical cores/threads, cache sizes, thermal design powers (TDPs), and transfer rates, see Appendix A3f. The power consumption between two processors is proportional when the frequency increases and the L2 / L3 cache size decreases at the same time. If the utilization levels are higher than 50 percent, the TDP and frequency influence increases. The power consumption exponentially increases when the utilization levels are larger than 80 percent and the frequency increases, as stated in [SIC 2003], see Equation (3.22). The TDP and frequency are negligible at lower utilization levels. If we compare processors with a 15-watt TDP difference, the power curves are approximately identical, and we observe the 15-watt gap between the various power curves when the utilization levels are larger than 80 percent. We estimate the power consumption on a linear basis for the processor, which has a few p-states. We adjust our calculation method for the processors with a higher amount of p-states towards a non-linear power curve. We empirically analyze the characteristics of the *Intel Xeon E5-2600v2* processor to define the effect on the slope $\Delta PO_{proc}$ and introduce the weight coefficients $WF_{P_k}^{proc}$ of our non-linear power correction $PC_{P_k}$ regarding the relevant characteristics, such as the number of p-states $k$, see Equation (5.77). We define the technical specification $CH_{TS}$ to consider the processor-specific characteristics regarding their generation or family.

$$PC_{P_k} = WF_{P_k}^{proc} * k * \Delta PO_{proc} \tag{5.77}$$

Furthermore, we found that the power curve is linear for processors with less than 15 p-states and non-linear for processors with more than 15 p-states, especially when the utilization level ($u_{proc}$) is between 20 and 80 percent. We specify a non-linear correction interval at the two edge regions of the power curve, firstly when the processor has a utilization level less than 20 percent and secondly, when the processor is at a high utilization phase ($u_{proc} \geq 80$), see Equation (5.78). The boundaries are specific to the *Intel Xeon E5-2600v2* generation, which we change in a flexible manner when we vary the processor. We separate the weight coefficients into $WF_{P_k}^{low}$ / $WF_{P_k}^{high}$ and found that the slope of higher utilization levels is always larger than the slope at lower utilization levels $WF_{P_k}^{high} \gg WF_{P_k}^{low}$. We consider the processors with less than ten p-states by a smaller weight coefficient $WF_{P_k}$, which is steady. We do not evaluate all possible characteristics that may enhance the power correction by $WF_{P_k}^{*}$ and neglect them, because of their insignificance. Our findings and concrete values are specific to the *Intel Xeon E5-2600v2* processor family, which need to be adjusted whenever we consider another processor.

$$WF_{P_k}^{proc} = \begin{cases} WF_{P_k}, & if\ k < 10 \\ WF_{P_k}^{low}, & if\ k > 15,\ u_{proc} \leq 20 \\ WF_{P_k}^{high}, & if\ k > 15,\ u_{proc} \geq 80 \\ WF_{P_k}^{*}, & otherwise \end{cases} \tag{5.78}$$

The default value of the $WF_{P_k}^{proc}$ is one that does not affect the power correction when we cannot define specific conditions. An exception is given by the thermal design power that defines the worst-case power consumption at the full utilization level. We do not adjust the TDP value in Equation (5.76) by a power correction $PC_{P_k}$. We found that the maximal power significantly relies upon the cache size and the processor threads. Both characteristics lead to the refinement of the worst-case power calculation. We specify the $PO_{proc}(100\%)$ by an additional weight coefficient $PC_{P_1}$, see Equation (5.79).

$$PO_{proc} = [TDP * PC_{P_1}, TDP - PC_{P_2}, TDP - PC_{P_3}, \dots, TDP - PC_{P_k}] \tag{5.79}$$

We calculate the maximal power, in accordance to the *Intel Xeon* architecture by a weight coefficient of 0.91. We determine two special cases, wherein we change the weight coefficients, as shown in Equation (5.80). In the case that the processor has a) the level two cache size ($L2$) bigger than $12x256KB$, b) a level three cache size ($L3$) larger than $30MB$, and c) more than 24 threads, we specify the power corrections by an increase of approximately four percent. In the second case, when a) the L2 cache size is smaller than $4x256KB$, b) the L3 cache size is smaller than $15MB$, and c) not turbo mode is available, we nearly halve the worst-case power by a weight coefficient of $0.54$.

$$PC_{P_1} = \begin{cases} 1.04, & if\ L2\ cache \geq 12x256KB, L3\ cache \geq 30MB, threads \geq 24 \\ 0.54, & if\ L2\ cache \leq\ \ 4x256KB, L3\ cache \leq 15MB, no\ turbo \\ 0.91, & otherwise \end{cases} \tag{5.80}$$

We do not concentrate only on the dynamic characteristics, because the static characteristics, such as the cache size, are significant characteristics when we model a next-generation component based on the basis of a predecessor. In common, we propose a worst-case power estimation, including the power correction $PC_{P_1}$, which uses a set of weight coefficients $WF = \{WF_{P_1}^1, WF_{P_1}^2, \dots, WF_{P_1}^l\}$ at the lowest p-state $P_1$ under the set of conditions $CN = \{CN_1, CN_2, \dots, CN_l\}$, whereby the conditions are a subset of any possible characteristics, see Equation (5.81).

$$PC_{P_1} = \begin{cases} WF_{P_1}^1, & if\ CN_1 \\ WF_{P_1}^2, & if\ CN_2 \\ \vdots & \\ WF_{P_1}^l, & if\ CN_l \\ WF_{P_1}^*, & otherwise \end{cases} \tag{5.81}$$

A shrunk die includes more registers on the same space; as a result, a core consumes more power than its predecessor generation, but provides more performance. The architecture limits the maximal power consumption because the cache size, core, and uncore area on the die differs[224]. We distinguish between the processor family and generation[225], which both specify the hardware configuration, but the family defines the server type, construction, the maximal amount of components, or the mounted motherboard. The generation limits the system-board components and the related series. We consider the characteristics presented in the previous sections as being typical of a wide range of the processors on the architectural level, whereby we need to adjust the weight coefficients for each of them. In general, we define a set of weight coefficients $WF = \{WF_1, WF_2, \dots, WF_l\}$ of each condition $CN = \{CN_1, CN_2, \dots, CN_l\}$, whereby the conditions are a subset of possible characteristics.

$$WF_{P_k}^{proc} = \begin{cases} WF_1,\ if\ CN_1 \\ WF_2,\ if\ CN_2 \\ \vdots \\ WF_l,\ if\ CN_l \\ WF_{P_k}^*,\ otherwise \end{cases} \tag{5.82}$$

In the next sections, we compare our approach to the academic processor power models and their corresponding characteristics, which address adequate methods of the specific server systems. The approaches concentrate upon a certain aspect of a particular processor and primarily characterize the processor power considering the dynamic characteristics. The power model $P_{proc}$ in [KJC et al. 2014] distinguishes into the core-based power, static power, and cache power, see Equation (5.83). [Bel 2001] states that the static processor power depends upon the time, voltage, and semiconductor characteristics.

$$P_{proc} = P_{dynamic} * cores + P_{static} + P_{cache} \tag{5.83}$$

According to [RRK 2008], the core-based power consumption is not a linear function of the utilization levels, because of the different number of active cores and shared resources. [BGM et al. 2010] and [BM 2012] present a core-based power model which sums the power consumptions of each core $k$, as shown in Equation (3.31).

$$P_l = \sum_{k=1}^{l} P_{core}(k) \tag{5.84}$$

We consider the number of active and available cores of each processor family and define the corresponding weight coefficients. Today's processors cannot change the core's frequency independent from each other, because the physical cores limit the base frequency. [GFN et al. 2006] presents that each component has diverse static and dynamic characteristics, whereby the voltage and frequency are the major characteristics. Bellosa [Bel 2001] shows that the dynamic power correlates to "the switching frequency of the transistors and size of the capacitors." The authors of [SIC 2003, JGM 2003, KJC et al. 2014] state that the processor

[224] Die differences: Haswell processor http://ark.intel.com/products/codename/42174/Haswell#@All, Nehalem processor http://ark.intel.com/products/codename/64237/Nehalem-EP#@All
[225] Processor: e.g., architecture (*Intel Core i3*), generation (*Ivy Bridge*), family (*i3-3xxx / i3-6xxx*)

frequency and capacitance are relevant characteristics, which we also consider in our concept. We execute two threads on a logical core of the same processor, which both fully utilize the core. The switching activities of the various functional units result in different power consumptions while executing diverse operations. We neglect this accuracy level to be independent from the architecture and the explicit workload. In [BGM et al. 2010], the authors define the core power by the access rates that correspond to the known event metrics. The authors of [Han 2007] state "…, a hybrid model of event counters and measured power may provide information to more fully describe the relationship between p-state, workload, and power consumption." The authors of [TMW et al. 1996] analyze a certain processor, including its internal behavior, and design the micro-architectural structure. The authors consider hardware-specific performance counters, as analyzed in [Bel 2000, SBM 2009, RAK et al. 2013, LSQ et al. 2014], which are unsuitable for novel hardware architectures because the performance counter differs between the generations. Another problem in the early design stage is that the hardware is not available. We avoid concrete hardware-specific event metrics in order to conceive a generic model and use state-based models. We abstract the micro-architectural structure to be independent of the exact processor. We cannot use the instruction-based approach proposed in [TMW et al. 1996], because we do not have any knowledge about the explicit instructions of every executed workload. Furthermore, the processor architecture restricts the instruction types, which we adjust the certain processors. The instruction types and operands differ, which results in a variable amount of clock cycles. The power model of [LJ 2003] addresses OS routine calls considering the instructions per cycle (IPC) and coefficients $K_0, K_1$, see Equation (5.85). The interrupts, processes, and inter-process controls are hardware-specific.

$$P = K_1 * IPC + K_0 \tag{5.85}$$

We cannot apply the approach of [LJ 2003], because we do not have any knowledge of the exact operating system and the OS routine calls of the server system while executing the application software. The power model in [Riv 2008] considers the relative processor utilization level $\frac{u_{proc}}{\max(u_{proc})}$, the coefficients $K_0, K_1$, and an empirical factor $F$, see Equation (5.86), to cover the processor behavior; whereby $F$ depends upon the processor characteristics, and $K_0, K_1$ relies upon the specific workload.

$$P = K_0 \mp K_1 * \left\{\frac{u_{proc}}{\max(u_{proc})}\right\}^1 \mp K_1 * \left\{\frac{u_{proc}}{\max(u_{proc})}\right\}^F \tag{5.86}$$

In principle, we refine the concept of [Riv 2008, RRK 2008] in the manner that we categorize, classify, and characterize each component of the entire system that specifies the empirical factor, e.g., the processor generation. We define the characteristics and extend the processor specification to replace the empirical factor. The authors in [IM 2003, BGM et al. 2010] propose an architectural scaling model considering the utilization levels, whereby each resource-bounded workload is weighted. In contrast, we consider the architectural scaling within our component models in the technical specification and configurations, which weights the relevant characteristics. We are aware of component-bounded workloads. The authors in [TDM 2011, MAC et al. 2011] consider the characteristics' generation, family, architecture, and technology. Table 28 and Table 29 conclude the common academic approaches and what characteristics the authors consider. It should be mentioned that this table is not all embracing, but is concentrating on the major characteristics that are significant for the processor power consumption.

Table 28: Processor characteristics $(I)$

| Processor characteristics | [TMW et al. 1996] | [Bel 2001] | [SIC 2003] | [JGM 2003] | [GFN et al. 2006] | [TDM 2011, MAC et al. 2011] | [BGM et al. 2010] | [BM 2012] | [KJC et al. 2014] | Our concept |
|---|---|---|---|---|---|---|---|---|---|---|
| **Static characteristics** | | | | | | | | | | |
| Cache / cache lines | | | | | | | | | y | y |
| Capacitors / capacitance | | y | y | y | | | | | | |
| Voltage | | y | | | y | | | | | y |
| Time | | y | | | | | | | | y |
| Semiconductor technology (TDP) | | y | | | | y | | | | y |
| Quantity (#) | | | | | | | | | | y |
| Status | | | | | | | | | | y |
| Type | | | | | | y | | | | y |
| Vendor | | | | | | | | | | y |
| Product life cycle stage | | | | | | | | | | y |
| Generation | | | | | | y | | | | y |
| Family | | | | | | y | | | | y |
| Series | | | | | | y | | | | y |
| Fabrication size (nm) | | | | | | y | | | | y |
| Resistance | | | | | | | | | | y |
| Performance features | | | | | | | | | | y |

Considered (y), not considered or unknown (no entry)

Table 29: Processor characteristics $(II)$

| Processor characteristics | [TMW et al. 1996] | [Bel 2001] | [SIC 2003] | [JGM 2003] | [GFN et al. 2006] | [TDM 2011, MAC et al. 2011] | [BGM et al. 2010] | [BM 2012] | [KJC et al. 2014] | Our concept |
|---|---|---|---|---|---|---|---|---|---|---|
| **Dynamic characteristics** | | | | | | | | | | |
| Cores / active cores (hyper-threading) | | | | | | | y | y | y | y |
| Frequency | | y | y | y | y | | | | | y |
| Accesses / instructions / operands | y | | | | | | y | y | | |
| Event metrics | y | | | | | | y | y | y | |
| Transfer rate | | | | | | | | | | y |
| Thresholds | | | | | | | | | | y |

Considered (y), not considered or unknown (no entry)

We concentrate upon the processor and memory characterization to develop an accurate power model. We neglect the I/O-based component characterization because we primarily specify the midrange server systems that execute processor-bounded and memory-bounded workloads.

**Fan Power Characterization Including the Thermal Models**

Besides the power consumption, the thermal development is a critical aspect of the energy efficiency. The components warm up and produce heat, which the thermal control mechanism cools down to avoid damage and ensure the working conditions. In the next sections, we describe the thermal models of the processor and memory, which we require to define the fan power and performance loss.

The package[226] heat dissipation, called the thermal resistance[227], is denoted by the theta $\theta_{res}$ or psi $\psi$ in the units of $[\frac{°C}{W}]$, which "…, indicates the steady-state temperature rise of the die junction above a given reference for each watt of power (heat) dissipated at the die surface. [Ben 2002]" The thermal resistance excludes the geometric effects [SXC et al. 2000], but differs

[226] Package: semiconductor device
[227] Thermal resistance: energy-to-temperature translation coefficient

into the thermal conductivity and electrical resistivity, including the material properties, as considered in the JEDEC-standard[228]. In the early design stages, the vendor rudimentary specifies the thermal characteristics of the entire system or the next-generation components. At first, we analyze the spreadsheets to specify the adequate thermal models. The components dissipate power and as a result, simultaneously produce heat.

We calculate the self-heating $\Delta T\ [°C]$ due to the power dissipation, which is a product of the thermal resistance $\theta_{res}$, supply voltage $V_{CN}$, and current $I_{CN}$ at certain conditions $CN$, as shown in Equation (5.87). In [NXP 2010], a condition of a memory module is the access on the serial data input/output (SDA) or occurring events.

$$\Delta T_{mem} = \theta_{res} * [V_{CN_1} * I_{CN_1} + V_{CN_2} * I_{CN_2} + \cdots + V_{CN_l} * I_{CN_l}] \tag{5.87}$$

In the case that we do not have any knowledge about the thermal resistance we define the thermal response on the basis of the self-heating, which is a step function response of a first-order system (PT1)[229]. We specify the memory thermal response $TH(t)_{mem}^{DE}$ considering an initial condition $CN_0$ and a time constant $T_S$. Equation (5.88) defines the temperature increase as an exponential function.

$$TH(t)_{mem}^{DE} = CN_0 * \left(1 - e^{-\frac{t}{T_S}}\right) \tag{5.88}$$

In contrast, we specify the processor thermal profile on the basis of linear relations, which we determine in the spreadsheets of existing processors. The vendor defines the thermal design power (TDP) and the corresponding processor case temperature $T_{CASE}$, which is maximal at the processor-specific TDP value. The minimal case temperature is a fixed operating temperature and specific to each processor family. The related power $P_{profile\_min}$ defines the lower power limit, as shown in the following equations.

$$(PO_2, TH_2) = \left(TDP, max(T_{CASE})\right) \tag{5.89}$$

$$(PO_1, TH_1) = \left(P_{profile\_min}, min(T_{CASE})\right) \tag{5.90}$$

[228] JEDEC: Joint Electron Device Engineering Council, https://www.jedec.org/

[229] First-order system or lag element (PT1): under certain conditions may be a second-order system (PT2)

Both critical data points define the thermal profile of a processor, which we specify as a linear equation, $TH_{proc}^{ST}$ including the slope $m_{TH}$ and the offset $n_{TH}$, as shown in Equation (5.91). The technical specification of an existing processor provides the values that we need to solve the equation. We assume that the thermal profile is always linear, as defined for a couple of processors, but the characteristics influence the slope or the offset. In the early design phase, the $T_{CASE}$ and $P_{profile\_min}$ are not available and, thus, we estimate both values by assuming the predecessor or a similar generation.

$$TH_{proc}^{ST} = m_{TH} * PO + n_{TH} \tag{5.91}$$

$$m_{TH} = \frac{\Delta TH}{\Delta PO} = \frac{TH_2 - TH_1}{PO_2 - PO_1}, n_{TH} = TH_2 - m_{TH} * PO_2 \tag{5.92}$$

In general, we characterize the temperature $TH_{proc}^{ST}$ as the power-related function, which is a common method of the thermal assumption for a server system within the data center [ERK 2006]. We transform the dissipated power $PO_{C_i}[W]$ into the temperature [°C] with the processor-specific weight coefficient[230] $WF_{C_i}^{TH}[\frac{°C}{W}]$, see Equation (5.93). In the case that our thermal models partly rely upon the power model, we consider the linear relation between the current and the ambient temperature, as done in [AR 2016]. The stationary temperature $TH_{C_i}^{ST}$ depends directly upon the ambient temperature and indirectly upon the thermal design power of a processor, which specifies the slope of the thermal profile. We extend the calculation by an offset $TH_{offset}$ to consider both influences.

$$TH_{C_i}^{ST} = PO_{C_i} * WF_{C_i}^{TH} + TH_{offset} \tag{5.93}$$

We define an exception of the component temperature especially in the case of the memory modules. The output current $I_{out}$ of the semiconductor devices decreases negatively exponentially by an increasing ambient temperature [SXC et al. 2000]. In contrast, the average supply current $I_{DD}$ of the memory modules increases linearly by the ambient temperature [NXP 2010]. The component temperature $TH_{C_i}$ behaves typically non-linear to the operating environment, such as the output current. [Han 2007] estimates the processor temperature by a linear regression considering the ambient temperature as an offset of the thermal specification. The authors argue that they can quickly calculate the processor temperature of all p-states and tolerate sensor delays. The approach does not consider the non-linear thermal development over the time. We define the stationary temperature as an initial condition of the thermal response $TH(t)$ and refine the step function response of a first-order system, see Equation (5.94).

$$TH(t)_{C_i}^{DE} = K_{C_i}^{ST} * \left(1 - e^{-\frac{t}{Ts}}\right) + TH_{C_i}^{ST} \tag{5.94}$$

[230] Weight coefficient: works, such as theta or psi, but is not defined by the vendor

In addition, we model the system fans to control the dynamic thermal development of the full system and consider the fan power, which is a static fraction at idle utilization levels, and a dynamic characteristic of the power when the temperature increases. We analyze the assembled server system fans and their characteristics. The authors of *BladeSim* [RL 2007], *SimOS* [RHW et al. 1995, Lan 2007], and *SoftWatt* [GSI et al. 2002] neglect the fan power.

In the approaches of [HS 2007, Han 2007, HKG et al. 2007], the authors define the component temperature by a certain state on the basis of the functional level, which we cannot estimate when we consider a flexible and generic workload. The authors in [APL et al. 2008] calculate the die temperature by the constant idle temperature and the dynamic temperature, which depends upon the performance counters of the hardware-specific events and the corresponding number of clock cycles while executing a software application. The authors of [SBA et al. 2011] present a thermal model to control the airflow within the server system with respect to the mounted memory modules. The concept analyzes the airflow and detects the hotspots within the system. The authors concentrate upon the computational fluid dynamics simulation (CFD), whereby the relevant characteristics are the system geometry, volume, air pressure, and humidity. The CFD in [LZZ et al. 2007, QXY 2008] requires a highly detailed thermal characterization, such as the resistors and capacitance architecture, which would increase the complexity of our entire system model, the simulated time, and development effort if we would apply this approach. We abstract the concrete airflow[231], but consider the stationary thermal development, e.g., the self-heating, and propose a simplified thermal model for each component development.

**Performance Models**

The performance model defines the real throughput under certain conditions. The academic approaches, such as [DEP et al. 2009] and [YZ 2011], execute diverse instruction types within benchmarks to study resource activities and get a relation between the functional units and their performance. We cannot predict the exact events or activities of our server system and therefore the instruction-based approach is not applicable. In [MKO et al. 2002], the authors observe specific events on the architectural level that rely upon the system family and generation. We do not investigate on the wide range of the performance models and their complexity. In general, we consider the same approach of the diverse performance models at various domains, which we include as a category-specific method on either the functional level or logical domain. We specify the relevant benchmarks and aggregate the performance metrics from multiple benchmarks in a database. We assume a linear relation between the utilization levels and the performance scores, which are maximal in the case of full utilization. The benchmarks provide only the largest scores[232], which we analyze in order to estimate the

[231] Airflow: including volume, pressure, airflow rate, and speed
[232] Scores: mean of performance ratios as a result of benchmarks

performance. We are not able to predict the explicit scores, because of our restricted knowledge about the exact workload or instructions. A single component $C_i$ may have several performance results because of the diverse benchmarks, as shown in Equation (5.95) for the *PassMark CPU* benchmark[233]. We consider the *Mixture* performance[234] as a default setting in our simulation model and abstract the specific processor instructions. The customer can adjust the simulation level of detail, such as selecting the *Floating Point Math*, to be more precise for a certain application software.

$$PE_{CPU} = Passmark\ CPU = \begin{cases} Mixture \\ Integer\ Math \\ Floating\ Point\ Math \\ Prime\ Numbers \\ Extended\ Instructions\ (SSE) \\ \dots \end{cases} \tag{5.95}$$

$$PE_{C_i} = \begin{cases} Passmark \\ SPEC \\ TPC \end{cases} \tag{5.96}$$

$$PE_{C_i} = \begin{cases} benchmark_1 \\ benchmark_2 \\ benchmark_{\dots} \\ benchmark_k \end{cases} \quad benchmark_k = \begin{cases} performance_{max}^{1} \\ performance_{max}^{2} \\ performance_{max}^{\dots} \\ performance_{max}^{p} \end{cases} \tag{5.97}$$

We neglect the time-based metrics, such as the response time, throughput, or bandwidth, which require a detailed model on the workload, executed operations[235], and the hardware architecture[236]. We consider the standard benchmark metrics, such as *PassMark*, *SPEC*[237], or *TPC*[238] instead. The common database stores many benchmark metrics of the diverse components $C_i$. If a customer-specific server configuration includes a component that is not part of our database, we parse the technical specification tree to find the closest relatives. We specify the weight coefficients concerning the performance at each level of the tree. We multiply the performance by the weight coefficients within the technical specification tree. In the case that we have to predict the future component performance, we define a performance boost, such as a 1.5 weight coefficient, between two different levels of the manufacturing technology.

233 PassMark: http://www.passmark.com/index.html
234 Mixture performance: also called *CPU Mark*, a mix of every defined benchmark within *Passmark CPU*
235 Workload and operations: determined by code inspection, input data set
236 Hardware architecture: memory hierarchy, compiler options, cache levels
237 SPEC: Standard Performance Evaluation Corporation, http://www.spec.org/
238 TPC: Transaction Processing Performance Council, http://www.tpc.org/default.asp

We update this coefficient when the next-generation results are available. We consider the specific benchmark results as a set of maximum scores that is reliable when the components have the highest utilization level. We adjust the scores when we reduce the utilization level down to idle analogous to the *SPEC*-benchmark performance scores.

In our concept, we transfer the performance results of the isolated components collected by the synthetic benchmarks to our system variations with the identical technical specification and configuration. If we simulate a component using the particular characteristics that we cannot reuse from the results, we update the weight coefficients to estimate the single component on the basis of its predecessor generation. We include the entire system measurements to cover the system-specific behavior of various server configurations. We combine the measurement results of the diverse synthetic benchmarks to estimate the behavior under certain conditions. In our simulation model, we reuse the benchmark results instead of developing the accurate performance models of each component or system variation. We specify a knowledge base that includes the component-based and system-based performance results.

**Classification as a Technical Assistant of the Category- and Aspect-based Characterization**

As stated in the previous sections, we found the diverse, relevant characteristics of a category $CS_i$ or a certain aspect $A_j$ which we include in our *logical and physical* configuration layer. We support a flexible characterization, which does not require the highest accuracy level. We found that the components $C_i$ in $CS$ have a similar characteristic and abstract them so that the models are independent of the concrete accuracy level, but consider sub-characteristics, in case we gained experience of the highest accuracy level. We adjust the concept of [GFN et al. 2006] and group our characteristics into shared classes $CL = \{CL_1, CL_2, \dots, CL_k\}$ to avoid redundant data of the various configurations. We define the common characteristics $CH = \{CH_1, CH_2, \dots, CH_k\}$, which stores the values of the explicit system configuration of a certain category. Figure 58 provides a brief overview of a memory module considering the system-board categories, their classification, and the related characteristics, which we describe in the next sections.

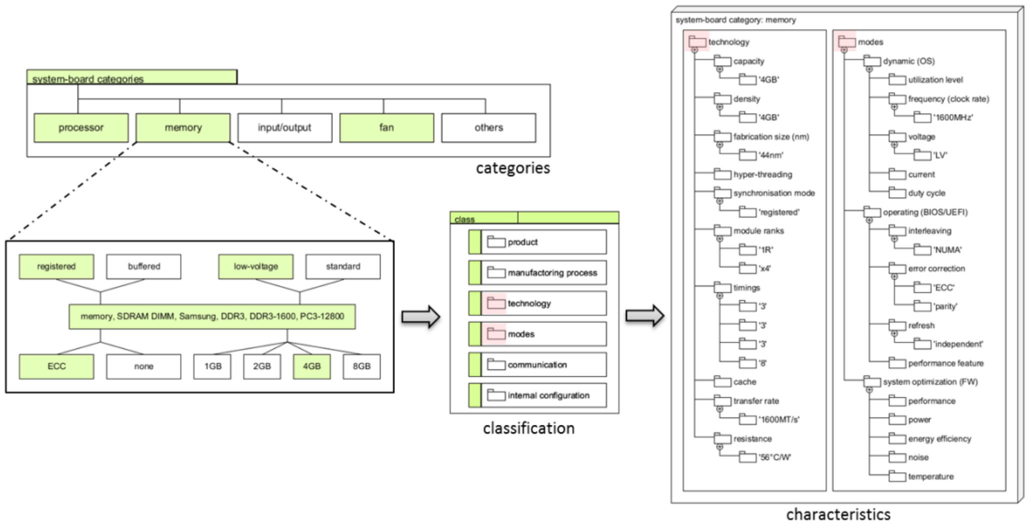


Figure 58: System categories, classes, characteristics, and values

We differentiate into a technical specification and a configuration, which may be static or dynamic. We define the classes as shown in Table 30 that helps to identify the dynamic characteristics, which we adjust during the simulation. On the other side, we adapt the technical specification, which may result in an extra simulation run. Herein, we differentiate in the *technology* and *manufacturing* changes that specify the component behavior. In contrast, we can change the component quantity, which influences the utilization level and respectively the power consumption.

Table 30: Class definition $CL_l$

| Terms $CL_l$ | Class | Tree | Configuration characteristics |
|---|---|---|---|
| $\boldsymbol{CL_1 = pro}$ | $product$ | $\theta$ | $CH_{TS}$ |
| $\boldsymbol{CL_2 = map}$ | $manufacturing\ process$ | $\theta_{TS}$ | $CH_{TS}$ |
| $\boldsymbol{CL_3 = tec}$ | $technology$ | $\theta_{TS}, \theta_{CS}$ | $CH_{CFG}^{ST}$ |
| $\boldsymbol{CL_4 = mod}$ | $modes$ | $\theta_{CS}$ | $CH_{CFG}^{DY}$ |
| $\boldsymbol{CL_5 = com}$ | $communication$ | $\theta$ | $CH_{TS}$ |
| $\boldsymbol{CL_6 = int}$ | $internal\ configuration$ | $\theta$ | $CH_{TS}$ |

The *product* class specifies the component quantity, if the component is active or still assembled in the system. Within the *product* class, the redundancy[239] characteristic is especially significant for the power supply and storage devices. We summarize the technical specification $CH_{TS}$ and the related characteristics in the *manufacturing process* class. We

[239] Redundancy: e.g., redundant array of independent disks (RAID), PSU redundancy $(N + 1, N + 2{,}2N, 2N + 1, N + N)$

define each component by its vendor, architecture, generation, family, and series. Herein, we assume the nomenclature of the *Intel* processors[240] and adapt them considering our components. A special characteristic within the *manufacturing process* is the product life cycle stage. For instance, *Intel* changes a pilot release processor during the production ramp-up. As a result, processors in the maturity, saturation, or decline stages have less manageable p-states[241] in comparison to the same product in the market introduction or growth stage. The vendor does not change the processor specification after a certain product life cycle stage. Therefore, the product life cycle stage is an empirical characteristic. The authors of [Han 2007] expect the increase of the individual component power and performance of the next-generation fabrication processes. We address Moore's law [Moo 1965] in the *manufacturing process* because we consider the technological development of the predecessor towards today's architecture, which results in a non-linear energy efficiency. Besides the technical specification, we include the static configuration $CH_{CFG}^{ST}$ of the on-chip technology, which we conclude in the *technology* class. The relevant processor characteristics are the hyper-threading, cache size, or the largest transfer rate, for instance. We consider the dynamic characteristics $CH_{CFG}^{DY}$ in our approach and differentiate the several component *modes*. First, the *dynamic* mode defines the real component state, such as the utilization level, frequency, voltage, current, or duty cycle. These characteristics are significant in the OS-specific management techniques, such as DVFS. Secondly, we sum up the BIOS/UEFI characteristics in the *operating* mode. Finally, the system *optimization* mode covers the firmware characteristics. We consider the I/O busses and connections within the *communication* class. The *internal* class defines the thermal thresholds, when a component is at the reliable, functional, or damage level, which is specific in each hardware configuration. We define an uncertainty or confidence level, which specifies the trust relation on the basis of the underlying measurement, spreadsheets, or simulations. We consider the classification in our technical specification tree and characterization tree. Figure 59 represents the *technology* and *modes* classes in the characterization tree of the $SDRAM\ DIMM, DDR3, DDR3-1600, PC3-12800$ memory module.

[240] *Intel* nomenclature: http://www.intel.com/content/www/us/en/processors/processor-numbers-data-center.html, http://ark.intel.com/

[241] Manageable p-states: formally, both processors have same state P0-P7, but P5 behaves like P6

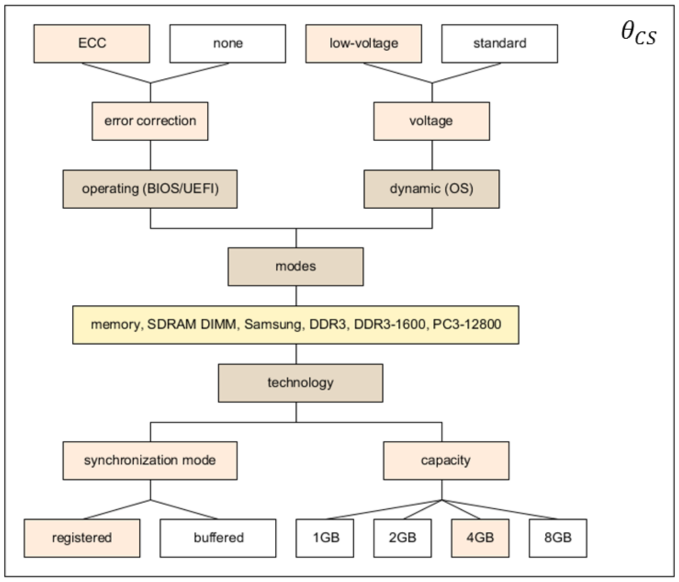


Figure 59: Characterization tree with classification

**Correlation of Component-based and Aspect-based Characterizations**

In the "*logical and physical* layer" section, we define the aspect-based component models regarding the server system configuration. We concentrate upon the relevant characteristics, values, and weight coefficients $WF^{CH}_{A_{j}C_{i}}$. We found that the aspects of a particular component influence each other. The processor dissipates the power $PO_{proc}(t_k)$, the temperature $TH_{proc}(t_k)$, and the performance $PE_{proc}(t_k)$ at a specific time $t_k$. The self-heating of the processor results in a temperature $TH_{proc}(t_{k+1})$, whereby the temperature increases $TH_{proc}(t_{k+1}) > TH_{proc}(t_k)$, but at the same time the performance decreases $PE_{proc}(t_{k+1}) < PE_{proc}(t_k)$. The authors of [NXP 2010, SXC et al. 2000] state that a larger temperature of semiconductor devices results in an increasing current, which limits the performance. The system fans compensate the warm-up process, which results in a higher power consumption of the system. Analogous to Newton's third law that every action has an opposite reaction, we apply the relevant principles concerning our aspects.

We consider the side effects of the certain components in the aspect-based relation $R_A$, as defined formally in Equation (5.31), which characterizes the mutual influences at a time $t_k$. We standardize the aspects $A_j$ towards their maximal possible values $max(A_j)$, which result in an interval $A_{j_{rel}}$ from zero to one. We specify the relation $R^{A_{k_{rel}}}_{A_{j_{rel}}}$ between the aspects as a sum of the relative values $A_{j_{rel}}$ and $A_{k_{rel}}$, which have to be within the closed interval $RR^{A_{k_{rel}}}_{A_{j_{rel}}}$ defined by the lower and upper limits $_{min}RR^{A_{k_{rel}}}_{A_{j_{rel}}}$ and $_{max}RR^{A_{k_{rel}}}_{A_{j_{rel}}}$, see Equation (5.100).

$$A_{j_{rel}} = \frac{actual(A_j)}{max(A_j)} \tag{5.98}$$

$$R_{A_{j_{rel}}}^{A_{k_{rel}}} = A_{j_{rel}} + A_{k_{rel}} \tag{5.99}$$

$$RR_{A_{j_{rel}}}^{A_{k_{rel}}} = \left[{}_{min}RR_{A_{j_{rel}}}^{A_{k_{rel}}}, {}_{max}RR_{A_{j_{rel}}}^{A_{k_{rel}}}\right], R_{A_{j_{rel}}}^{A_{k_{rel}}} \in RR_{A_{j_{rel}}}^{A_{k_{rel}}} \tag{5.100}$$

The limits cannot be accessed generically because another component characteristic $CH$ may restrict the maximal throughput, which results in a lower ${}_{max}RR_{A_{j_{rel}}}^{A_{k_{rel}}}$. Therefore, we extend our relations in $R_{A_{j_{rel}}}^{A_{k_{rel}}}$ concerning the various characteristics. Equation (5.102) formally shows the relation definition between power and performance. We specify the complete component behavior in the model $BE_C$ being aware of such feedback loops between the aspects.

$$\begin{gathered} {}^{CH_1}R_{PE_{rel}}^{PO_{rel}} = PO_{rel} + PE_{rel},\ {}^{CH_1}RR_{PE_{rel}}^{PO_{rel}} = \left[{}_{min}^{CH_1}RR_{PE_{rel}}^{PO_{rel}}, {}_{max}^{CH_1}RR_{PE_{rel}}^{PO_{rel}}\right] \\ {}^{CH_2}R_{PE_{rel}}^{PO_{rel}} = PO_{rel} + PE_{rel},\ {}^{CH_2}RR_{PE_{rel}}^{PO_{rel}} = \left[{}_{min}^{CH_2}RR_{PE_{rel}}^{PO_{rel}}, {}_{max}^{CH_2}RR_{PE_{rel}}^{PO_{rel}}\right] \\ \vdots \\ {}^{CH_l}R_{PE_{rel}}^{PO_{rel}} = PO_{rel} + PE_{rel},\ {}^{CH_l}RR_{PE_{rel}}^{PO_{rel}} = \left[{}_{min}^{CH_l}RR_{PE_{rel}}^{PO_{rel}}, {}_{max}^{CH_l}RR_{PE_{rel}}^{PO_{rel}}\right] \end{gathered} \tag{5.101}$$

$$R_{PE}^{PO} = \begin{cases} {}^{CH_1}R_{PE_{rel}}^{PO_{rel}} \in {}^{CH_1}RR_{PE_{rel}}^{PO_{rel}},\ if\ CH_1 \\ {}^{CH_2}R_{PE_{rel}}^{PO_{rel}} \in {}^{CH_2}RR_{PE_{rel}}^{PO_{rel}},\ if\ CH_2 \\ \vdots \\ {}^{CH_l}R_{PE_{rel}}^{PO_{rel}} \in {}^{CH_l}RR_{PE_{rel}}^{PO_{rel}},\ if\ CH_l \\ 1,\ otherwise \end{cases} \tag{5.102}$$

After defining the relations at a time $t_k$, we define the dynamic behavior between the time step $t_k$ and $t_{k+1}$ in the relation $R_{t_k}^{t_{k+1}}$. We consider the actual condition of the component, such as the increasing or decreasing relative values of each aspect $A_j$, and specify the impacts due to the remaining aspects, see Equation (5.103).

$${}_{A_j}R_{t_k}^{t_{k+1}} = \begin{cases} A_{j_{rel}}(t_{k+1}) > A_{j_{rel}}(t_k) \stackrel{\text{def}}{=} impacts \\ A_{j_{rel}}(t_{k+1}) < A_{j_{rel}}(t_k) \stackrel{\text{def}}{=} impacts \end{cases} \tag{5.103}$$

As an example, we assume the increasing power consumption $PO_{rel}(t_{k+1}) > PO_{rel}(t_k)$ as an aspect-based result of our calculation methods, which we specify as $PO_{INC}$. We define the impacts of the remaining aspects $TH$ and $PE$ by the weight coefficients $WF_{PO_{INC}}^{TH}$ and $WF_{PO_{INC}}^{PE}$, which we apply upon the relative aspects at the time step $t_k$, as shown in Equation (5.104).

$$\begin{aligned} TH_{rel}(t_{k+1}) &= TH_{rel}(t_k) * \ WF_{PO_{INC}}^{TH} \\ PE_{rel}(t_{k+1}) &= PE_{rel}(t_k) * \ WF_{PO_{INC}}^{PE} \end{aligned} \tag{5.104}$$

Another challenge is that the components themselves influence each other within the entire system. The authors in [YP 2009] state "…, the memory access is extremely costly in terms of the CPU core clock cycles and thus the memory system turns into the main bottleneck in system performance. This is mainly because the speed gap between the fast CPU core and the relatively slow memory widens." We specify dynamic rule tables in the matrix $R_{BE}$ to restrict the component interactions. The authors of [ERK 2006] measure the system components in isolation and do not cover system-specific effects. The concepts of [ERK 2006, Riv 2008, RRK 2008] consider the performance counters, which are unsuitable for novel systems because of their dependency to the architecture. Our aim is to reduce the extra modeling effort, while being independent of the explicit hardware configuration and characterization. [BHS 1998] state "When the CPU is active, the memory components are also active to some degree." The utilization levels vary across a time series on the basis of the complex system interactions on the busses and activities, which depend upon each other. We found that we could not fully utilize the processor and memory at the same time, which we specify as internal constraints. We restrict the maximum utilization levels, which the components may have at the same time. We exemplarily define the relation between the component $C_1 = mem$ and $C_2 = proc$, which can be done analogous to the aspect-based relation $R_A$. We specify the maximal utilization level ${}_{max}u_{proc}^{mem}$ that the sum of the memory and processor utilization level $(u_{mem} + u_{proc})$ cannot exceed. At the same time, the minimal utilization level ${}_{min}u_{proc}^{mem}$ has to be fulfilled. We specify the behavioral relations between the components in $R_{BE}$, respectively their utilization levels, which form the steady simulation constraints.

$$R_{proc}^{mem} = u_{mem} + u_{proc} \tag{5.105}$$

$$RR_{proc}^{mem} = \left[{}_{min}u_{proc}^{mem}, {}_{max}u_{proc}^{mem}\right], R_{proc}^{mem} \in RR_{proc}^{mem} \tag{5.106}$$

In the case that we define the relations in $R_{BE}$ and restrict the utilization levels, we indirectly limit the worst-case power consumption of the entire system. The full-system power model of [ERK 2006] addresses the system behavior considering an explicit operating system, including the particular design and architecture properties. The authors define the component-bounded utilization levels that correspond to the OS behavior, but do not address the details of the coefficients $K_0, K_1, K_2, K_3$, and $K_4$, see Equation (5.107). The authors change the coefficients $K_1$and $K_2$, which refers to the relation between the processor-bounded and memory-bounded workload at the OS. The system power depends upon the activity level and the communications, which enables various states and transitions.

$$P_{server} = K_0 + K_1 * u_{processor} \mp K_2 * u_{memory} + K_3 * u_{disk} + K_4 * u_{network} \tag{5.107}$$

In our concept, the coefficients do not only weight the utilization levels of each component; the coefficients also define the relation to each other referring to the technical specification and configuration of each component. We characterize the system design and architecture in an additional layer, which is a more precise in comparison to [ERK 2006] considering diverse component variation.

Furthermore, the authors in [YP 2009] describe, “Apparently, without memory involved, the CPU utilization should scale linearly with the CPU core frequency. For example, if the CPU utilization is 50% at 104MHz with little or even no memory access, we can easily predict that the CPU utilization would be around 25% at 208MHz with the high confidence. But with memory access involvement, the CPU utilization would not scale linearly with the core frequency any more. At the higher frequency point, performance is usually more blocked by the memory since the CPU spends more CPU cycles in waiting for memory response.” We cover the performance loss in the relations $R_A$ and $R_{BE}$, which define the influence between the components or one certain component. The previous academic results lead to define the relations on the component as well as the system level considering the certain workload. The workload specifies the processor power fraction of the total system consumption, which decreases because the memory modules consume more power in dependence on their shrinking technology in the novel generations [RRK 2008]. The memory power becomes more significant at the processor-bounded and memory-bounded workloads, when the system has a high number of memory modules, such as 64.

Other key indicators of the system behavior are the management techniques, especially of the power consumption. The techniques concentrate upon the average power to optimize the energy consumption over the time, which saves electricity costs. The peak power reduction decreases the cooling costs in the data centers, which we outline in Section 4.4. In the large-scale enterprises, the server system is limited by a power budget to ensure that the total power and the temperature do not reach a given boundary. The authors of [RLI et al. 2006] optimize the dynamic power budgeting at a blade enclosure, which reduces the power provisioning between the various server blades. The management technique shall provide the suitable power to the current demands. The management techniques are either a hardware solution, which has direct access to the low-level information, or a software solution, which operates at the application level. The time granularity in the hardware is based upon seconds or milliseconds, which is evolving towards hours in the software technique. A well-known power management technique is dynamic voltage frequency scaling (DVFS) in which the operating system controls the component states on the basis of the utilization level of scaling down the unused components, see Section 3.4.2. We propose an individual management technique considering the workload variation, while being aware of the high-activity ($u_{C_i} > 80\%$) and low-activity ($u_{C_i} < 20\%$) utilization phases.

According to the approach in [ZMC 2003], we save energy when we reduce the frequency $f$ or supply voltage $V_{DD}$ of the processor. The dynamic power consumption $P_{dynamic}$ is a product of the static capacitance $C_L$, the square supply voltage $V_{DD}$, and the frequency $f$.

$$P_{dynamic} = C_L * V_{DD}^2 * f \tag{5.108}$$

"When decreasing processor speed, we can also reduce the supply voltage. This reduces processor power cubically and energy quadratically at the expense of linearly increasing the task's latency. [ZMC 2003]" We assume that the frequency $f$ and performance, which we specify as execution time $T$, are inversely proportional to each other. If we restrict the actual frequency to half of it, the time $T_f$ will double to execute the same job. As an algorithm constraint, we have to guarantee that the allocated time $T_{max}$ is always long enough to finish the job when we reduce the supply power and frequency. In an ideal case, our management does not have any negative impact upon the system or component performance, such as the execution time.

$$\frac{f}{f_{max}} = \frac{T_{max}}{T_f} \tag{5.109}$$

The non-linear power consumptions of the processor lead us to adjust the frequency, as long as we satisfy the time demand $T_f \leq T_{max}$. We alter the frequency $f$ and voltage $V_{DD}$ of a processor in the intervals $f_{min} \leq f \leq f_{max}$, ${V_{DD}}_{min} \leq V_{DD} \leq {V_{DD}}_{max}$, which the customer specifies implicitly in the server system configuration $\theta_C$. We do not toggle between the various frequencies within a time step, to avoid the additional transition times. We search the local optimum to minimize the power consumption and corresponding temperature, but try to find the maximal performance at the same time. This is a well-known resource management technique, called DVFS, which controls the power consumption in a time horizon of seconds. Our aim is to optimize the energy efficiency ratio in a time horizon from seconds to hours or days. The common management techniques concentrate upon the actual states and power consumption, which we include by calculating the corresponding aspects at each time step. The management techniques correlate to the component-based and aspect-related characterizations of the explicit server configuration $\theta_C$, which may limit the opportunities. We apply the DVFS technique in our processor model to reduce the average power consumption.

Our aim is to optimize the energy efficiency of the server system $SY$ considering all components in $\theta_C$ executing a customer-based application software $W$. The static part of our system considers the architecture and connectors, which rely upon the server system configuration. We consistently consider the same architecture because we support only a subset of the server systems in our simulation model. We neglect the static configurations $AC(\theta_C)$ and $CC(\theta_C)$, which we cannot adjust in our prototype implementation. The dynamic part considers the system behavior, its components, and relations to each other, see Equations (5.112) and (5.113).

$$SY = \{ST_s(\theta_C), DY_s(W, \theta_C)\} \tag{5.110}$$

$$ST_s(\theta_C) = \{AC(\theta_C), CC(\theta_C)\} \tag{5.111}$$

$$DY_s(W, \theta_C) = \{BE_C(W, \theta_C), R_{BE}(W, \theta_C)\} \tag{5.112}$$

$$= \{MAS_C(W, \theta_C), R_{A_C}(W, \theta_C), R_{BE}(W, \theta_C)\} \tag{5.113}$$

The relations $R_{A_C}$ and $R_{BE}$ depend more upon the workload $W$ than on the customer-specific configuration $\theta_C$. The power-to-temperature impact is similar between two components of the same category, but not comparable when executing different workloads $(W^1, W^2)$. We use the same assumption for the relation $R_{BE}$. We sum up the system in Equation (5.117).

$$R_{A_C}(\theta_C^1) \sim R_{A_C}(\theta_C^2),\ R_{A_C}(W^1) \neq R_{A_C}(W^2) \tag{5.114}$$

$$R_{BE}(\theta_C^1) \sim R_{BE}(\theta_C^2),\ R_{BE}(W^1) \neq R_{BE}(W^2) \tag{5.115}$$

$$DY_s(W, \theta_C) = \{MAS_C(W, \theta_C), R_{A_C}(W), R_{BE}(W)\} \tag{5.116}$$

$$SY(W, \theta_C) = \{MAS_C(W, \theta_C), R_{A_C}(W), R_{BE}(W)\} \tag{5.117}$$

We cannot optimize the aspect-based relations[242] $R_{A_C}(W)$ within one component or the relationships $R_{BE}(W)$ between the diverse components separately, because both depend upon the workload scenario of the customer and influence each other. We specify a relation by the weight coefficient, which defines the sensitivity[243] between two tending variables. In our optimization strategy, we concentrate on the aspect-based component models in the matrix $MAS_C(W, \theta_C)$.

$$MAS_C(W, \theta_C) = \begin{pmatrix} PO_{proc} & PO_{mem} & PO_{io} & PO_{fan} & PO_{oth} \\ PE_{proc} & PE_{mem} & PE_{io} & PE_{fan} & PE_{oth} \\ TH_{proc} & TH_{mem} & TH_{io} & TH_{fan} & TH_{oth} \end{pmatrix} \tag{5.118}$$

We simultaneously optimize $k$ $(k \geq 2)$ objective functions $F(x)$, which we define as the aspect-based calculation methods $F_{A_{j_{C_i}}}$ in the matrix $MAS_C$. Each function depends upon the workload $W$ and the server system configuration $\theta_C$. We run into a multi-objective optimization problem (MOP) because of competing methods.

$$F(x) = F(W, \theta_C) = MAS_C(W, \theta_C) = \{F_{A_{j_{C_i}}}(W, \theta_C)\} \tag{5.119}$$

242 Relations: effect on each other, defined without any unit
243 Sensitivity: strength of dependence

The functions $F_{A_{j_{C_i}}}$ in the matrix $MAS_C$ consist of the technical specification functions $F_{TS}$ and the configuration functions $F_{CFG}$ to calculate the adequate fraction. We neglect the workload as an input parameter in the technical specification functions because the technology impact influences the components at all utilization levels in the same manner. We divide the configuration function $F_{CFG}$ into the static $F_{CFG}^{ST}$ and dynamic $F_{CFG}^{DY}$ constituents, see Equation (5.126).

$$F_{A_{j_{C_i}}}(W,\theta_C) = F_{TS}(\theta_C) + F_{CFG}(W,\theta_C) \tag{5.120}$$

$$F_{CFG} = \{F_{CFG}^{ST}, F_{CFG}^{DY}\} \tag{5.121}$$

Each of these functions can be defined by the component-specific characterstics $CH$ of $\theta_C$, which we weight by the corresponding coefficients $WF$, and consider the optional offsets $OF$. The simulation model of each component $A_{j_{C_i}}$ specifies the relevant characteristics of the diverse classes for the calculation methods in $F_{A_{j_{C_i}}}$. We consider the technical characteristics of $F_{TS}$ in the *manufacturing process* class and include the static configuration characteristics $CH_{CFG}^{ST}$ in the *technology* class. We cover the characteristics of the DVFS and DTM techniques by the *modes* class, which specify the dynamic characteristics $CH_{CFG}^{DY}$ in our approach. The following equations show a simplified set of the objective functions concerning the characteristics. We describe the concrete characteristics of each component in the implementation section.

$$F_{TS} = CH_{TS} * WF_{TS} + OF_{TS} \tag{5.122}$$

$$F_{CFG}^{ST} = CH_{CFG}^{ST} * WF_{CFG}^{ST} + OF_{CFG}^{ST} \tag{5.123}$$

$$F_{CFG}^{DY} = CH_{CFG}^{DY} * WF_{CFG}^{DY} + OF_{CFG}^{DY} \tag{5.124}$$

We specify a vector of $k \in N$ decision variables $x = (x_1, x_2, \dots, x_k)$, also called parameters, which belong to the feasible region – a set of solutions – that satisfies at least one given objective function, either minimize or maximize $F_{A_{j_{C_i}}}(W,x)$ of $MAS_C$. We assume that the workload $W$ is steady for a simulation run. The decision variables affect the server configuration $\theta_C$, which automatically results in the adjustments of its corresponding technical specifications, static, and dynamic characteristics.

$$x = \{\theta_C, CH_{TS}, CH_{CFG}^{ST}, CH_{CFG}^{DY}\} \tag{5.125}$$

We optimize the energy efficiency $(EE)$ of the entire server system considering the decision variables $x$, which we formally specify as the optimization aim $max\{F(x)\}$. We formally define $F(x) = \big(F_1(x), F_2(x), \dots, F_k(x)\big)$, which contains $k \in N$ multi-objective functions. In our concept, the energy efficiency calculation considers the power, temperature, and performance methods[244]. We define the energy efficiency as the ratio between the performance $PE$ and power $PO$ over a period of time $T$. We assume that this period of time $T$ is constant in our optimization, which we cannot change. Our aim is to optimize the performance-to-power ratio, which is highest when we maximize the performance $max(PE)$, and at the same time minimize the power $min(PO)$, see Equation (5.126). We minimize the thermal aspect $min(TH)$ as a consequence of the power optimization, which is based upon each other. We consider the entire server system and therefore optimize all components.

$$max(EE) = max\left(\frac{PE}{T*PO}\right) \stackrel{\text{def}}{=} max\left(\frac{PE}{PO}\right) = \frac{max(PE)}{min(PO)} \tag{5.126}$$

We define a set of constraints $G(x)$ that further restricts the results of each function in $F(x)$. In our server system, we limit the minimal performance provided by the initial customer configuration[245] and neglect the solutions with less performance. We avoid higher power values than the customer-specific components consume, which are upper constraints of the power-based methods. In addition, we specify a lower as well as an upper bound $x^L, x^U$ of each variable to consider the hardware[246] / software[247] constraints on the exact configuration.

$$subject\ to: G(x) = \big(G_1(x), G_2(x), \dots, G_l(x)\big) \geq 0, l \in N\ and\ x^L \leq x \leq x^U \tag{5.127}$$

In our concept, the performance-to-power ratio[248] of each component cannot be less than zero because we normalize the power and performance, see Equation (5.128). If the power consumption is very small, the performance-to-power ratio increases arbitrarily highly. We normalize the ratio $G_1^C(EE)$ in the range to [0,1]. Therefore, we define the maximum of the performance-to-power ratio by the minimal power consumption ($PO_{C_{idle}}$), see Equation (5.129), which forms the upper limit of the range. We normalize the values of $G_1^C(EE)$ considering the formula in Equation (5.130).

$$G_1^C(EE) = \frac{PE_C}{PO_C} \geq 0,\ \ 1 \geq PO_C > 0,\ \ 1 \geq PE_C > 0 \tag{5.128}$$

$$\max\left(G_1^C(EE)\right) = \frac{\max(PE_C)}{\min(PO_C)} = \frac{1}{PO_{C_{idle}}}, \min\left(G_1^C(EE)\right) = 0 \tag{5.129}$$

$$normalized\left\{G_1^C(EE)\right\} = \frac{G_1^C(EE) - \min(G_1^C(EE))}{\max\left(G_1^C(EE)\right) - \min(G_1^C(EE))} = \frac{G_1^C(EE)}{\max\left(G_1^C(EE)\right)} \tag{5.130}$$

[244] Energy efficiency calculation: related functions $F_{A_{jC_i}}(W, x)$ of $MAS_C$

[245] Initial customer configuration: $\theta_C$ calculated $EE_{BASE}$

[246] Hardware constraints: a (physical) limit, such as the maximal frequency

[247] Software constraints: the temperature limits (e.g., TCONTROL, PROCHOT, THERMTRIP)

[248] Ratio: division of normalized performance scores and normalized power values

We follow the alternation strategy in a hierarchical manner when adjusting the decision variable $x$, which has to be system-compatible. We consider the aspect-based relations $R_{A_C}(W)$ within one component or the relationships $R_{BE}(W)$ between the diverse components when maximizing the energy efficiency of the entire system. We implicitly define further constraints in the aspect-based calculation methods, such as non-linear relations between the quantity of components and their performance. While optimizing the server system, we make sure that the thermal limits are respected. We adjust the set of decision variables $x$ that result in $\breve{x}$ to find an ideal solution. We apply the changes on the basis of the main principles of our heuristic methods, see Section 5.2.1 and 5.4.2.2. We determine the particular set of values $\breve{x} = (\breve{x}_1, \breve{x}_2, \dots, \breve{x}_k)$ among the set of all objective functions, which satisfy the constraints and conditions. A single optimal solution (global optimum) is usually not possible, which means that the results may be a corridor of local optimums instead. Therefore, we define a *Pareto front* as a set of solutions that partly satisfy the objective functions. We specify an optimization strategy in Section 5.4.2.1 to change the decision variables.

The authors of [Han 2007] concentrate upon the coordinated power, energy, and performance management of a particular system. The pipeline throttling, DVFS, and cache sleep methods build the base of their management technique. In next-generation server systems, an extra controller mounted on the motherboard scales the voltage and frequency autonomously. The OS loses the frequency control, and thus the OS type becomes negligible in such novel systems. Certainly, the vendor can specify the management techniques and their constraints independently of the OS.

The dynamic thermal management (DTM) is another technique to reduce the cooling costs of a time horizon of seconds and minutes. The aim is to reduce the temperature of certain components or the entire system. The fan control system aggregates the component temperatures[249], which forms the basis of the fan speed control (FSC) algorithm. We control the active cooling devices (fans) by their discrete speed $[RPM]$ using either a linear algorithm or table-driven (step-based) approach, as shown in Figure 60. The linear algorithm controls the fan speed[250] $[\%]$, which increases proportionally between the minimal and maximal temperatures. A vendor-specific constraint is the minimal fan speed $[\%\ or\ RPM1]$, which can be zero when energy-efficient components are assembled in the server system. The table-driven approach considers a set of temperatures $T = \{T_1, T_2, \dots, T_k\}$, which defines the corresponding set of RPM values $RPM = \{RPM_1, RPM_2, \dots, RPM_k\}$. The step-based approach provides the possibility to control the fan speed on a non-linear basis. The vendor specifies the thermal design before the server system is introduced to the market.

[249] Component temperatures: analog or digital sensors, accuracy varies from $1°C$ to $3°C$

[250] Fan speed: percent of duty cycle, which adjusts the pulse width modulated (PWM) signal to control the speed $[RPM]$

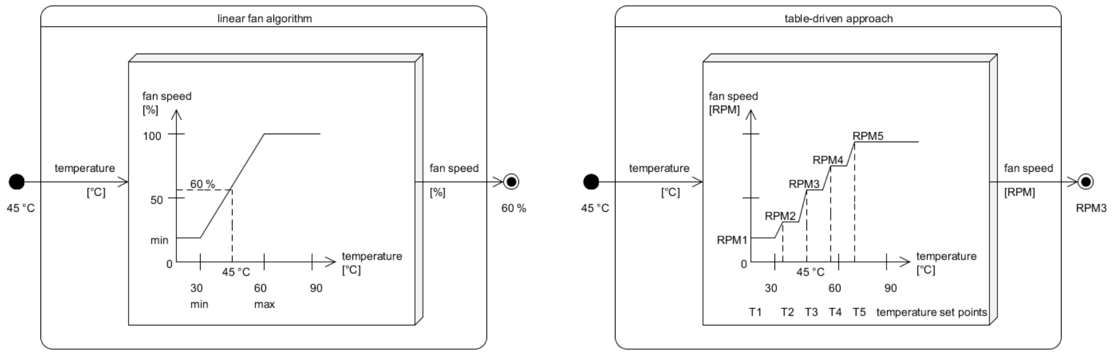


Figure 60: Fan speed control (FSC) algorithm

In our concept, we only consider the linear fan algorithm. We assume a closed-loop fan speed control because the fans have their own logic to ensure that target value will not be exceeded. We consider the thermal limits of the system where the components will be able to work, as shown in Figure 36. We include the temperature range and update our fan algorithm. We outline our overall control algorithm in Section 5.4.

## 5.2.3 Configuration – *Process and Control* Layer $SY(\theta_C, \delta, \upsilon, \chi)$

### *5.2.3.1 Process Definition*

The *process* layer combines the physical system environment, which we define as externals $EX$, and the specific server system $SY$. We consider the *logical and physical* description of the server in the *characterization* layer. The *process and control* layer defines the communication [251] between the components considering the connectors $CC$ and the architectural description $AC$ of the *configuration* layer. The workload characterizes the executions of the operations and the related communications, which affects the component activities and interactions. An input/output-bounded workload produces more network communication in comparison to a processor-intensive workload, which utilizes the bus between the processors and memories. The processor has to wait until the memory accesses finish. A shared cache across the multiple processors will reduce the communications via the bus. Furthermore, the associated memory banks of the processor influence the performance in dependence on the interleaves and bus connections. We differentiate the memory accesses if we utilize the memory communication when the system has one or more processors. The authors in [Bel 2001] state that the communications through the bus systems contribute the main dynamic energy consumption in dependence on the cycle-based load/unload scenarios of the registers.

[251] Communication: characteristics defined in the *communication* class

The *process* layer specifies the dynamic system behavior $DY_s$ and distributes the communication $\delta$ within the entire server system. According to the static system definition, we specify the category-specific utilization levels as an input parameter of the aspect-based calculation methods. We define a set of generic category-specific interfaces considering the flexible amount of components. In the case that the system has one memory module, we do not adjust the memory-bounded utilization level, as exemplarily defined with a value of 50 percent. Our model considers the hardware alternation, such as we assemble four memory modules. We assume any combination of the component-specific utilization levels, which results in a mean value of approximately 50 percent: e.g., we utilize all modules at nearly 50 percent, fully utilize half of the modules, or specify any utilization levels in between. The customer specifies the utilization level $u_{C_i}$ of each component $C_i$ as an input parameter of our simulation model, see Section 5.3.2.2. We predefine $k, k \in N$ profiles to simulate the workload distribution in a flexible manner. The customer can extend the profiles or select a certain profile, which matches the reality most probably. We require these profiles when we alter the hardware configuration by the amount of components within the product class. We implement $k \in \{1,2,3\}$ profiles within our simulation model, which supports the flexible configuration of the workload. In the first profile, we consider the initial utilization level $u_{C_i}$ and provide the same value over all components $p$ of ${C_i}^p$ as a default simulation parameter. This profile is common in the industrial tools. In the second profile, we support the component-bounded workload, which refers to the academic approaches. The third profile defines a uniform distribution among the components of the same category depending upon the component's amount.

$$u_{C_i} = \begin{cases} profile_1 \\ profile_2 \\ \vdots \\ profile_k \end{cases}, if\ \#C_i > 1 \tag{5.131}$$

$$C_i = \{{C_i}^1, {C_i}^2, \dots, {C_i}^p\}, p \in N,\ \#C_i = |C_i| = p \tag{5.132}$$

$$profile_1 \equiv u_{C_i} = u_{C_i^1} = u_{C_i^2} = \cdots = u_{C_i^p} \tag{5.133}$$

$$profile_2 \equiv \begin{cases} u_{C_i} = u_{C_i^1} \\ u_{C_i^2} = \cdots = u_{C_i^p} = 0 \end{cases}, if\ select\ C_i^1 \tag{5.134}$$

$$profile_3 \equiv u_{C_i^1} = u_{C_i^2} = \cdots = u_{C_i^p} = \frac{u_{C_i}}{p} \tag{5.135}$$

Within the *controller* layer, we observe the power, temperature, and performance of each component and store the results $\varepsilon$ during the simulation. Herein, we generate reports, display graphs, and present the energy efficiency ratio in relation to the optimal server configuration. The controller checks the workload and communication for compliance with the constraints that we specify in the *characterization* layer. We monitor the utilization levels of the components, for example, to ensure the defined relationships $R_{BE}$. If the customer selects a profile, the simulation parameters are affected, which we have to adjust. We consider the external constraints, such as $\alpha_{volt}, \theta_c$, the internal constraints $\upsilon$, such as $R_{BE}, R_{PE}^{PO}$, and the resulting restrictions, such as $f_{min}, f_{max}$, towards our simulation constraints $\chi$. The *process and control* layer enables the opportunity to perform multiple simulation runs by managing parameters and constraints and by applying the results of our optimization algorithm. We validate the aspect-based data considering the values in our database. In the case of the processor performance, we make sure that the actual value is higher than the peak performance measured in a synthetic benchmark. We define an uncertainty or confidence level by a scalar value that specifies the trust relation in dependence on the underlying real measurements, spreadsheets, empirical data, or simulation runs. The value is within the interval $[0,100]$, whereby a low value refers to an initial estimation and the highest value presents a certified result. We check the correctness, consistency, and plausibility using the *internal* class and database.

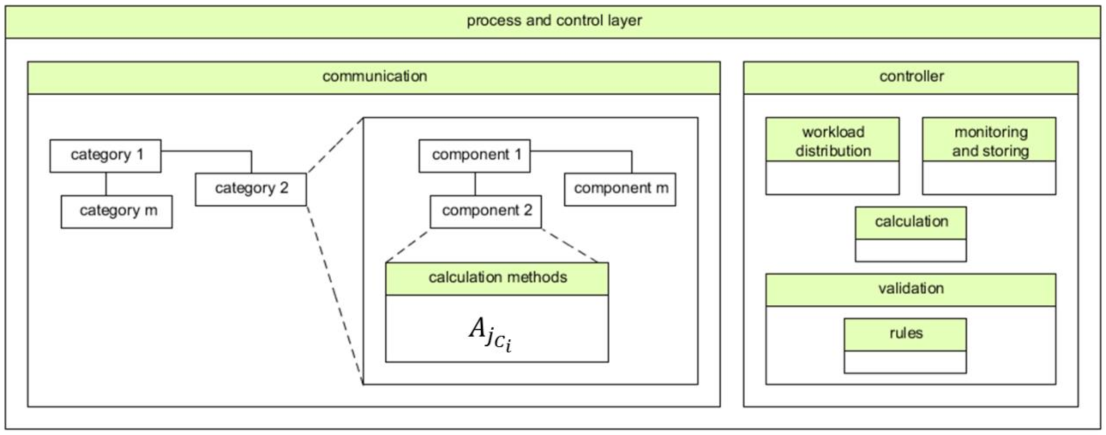


Figure 61: Process and control layer

The server system *configuration and characterization* layer deploys the hardware architecture, generation, and related components. Herein, we consider the aspect-based models in $MAS_C$ as a white-box, gray-box, or black-box approach[252] from a lower level to an upper level, but relying on the individual utilization levels. We specify the corresponding linear and non-linear behavior of every component, provide interfaces to include the academic approaches to the instruction level, and extend our observations in the industrial practice of accurate results in academia. We decouple each component model and define the behavior separately because of its reusability for next-generation systems, which reduces the modeling effort. Our hierarchical concept supports the extension of the components at various abstraction levels, whereby we provide the category-specific interfaces. We specify the characteristics across

[252] Approach: based upon the level of details

multiple components and cover both inter- and intra-component communication in the *process and control* layer, which enables optimization strategies across various layers within a server system. The *server system configuration and characterization* define the conceptual, contextual, and mathematical models, such as:

- The server system and its components
- The relations between the components
- The categories, classes, and characteristics
- The differential equations to calculate the aspects
- The generic configuration tree
- The relations between the aspects
- The customization parameter of the simulation model

## 5.3 Server System Externals $EX = \{\alpha, \eta, W, \xi, T_u\}$

We formally define the server system itself and its components, as shown in the following equation. In this section, we concentrate upon the externals $EX$, which specifies the physical environment of the entire server system.

$$[EX]\underset{\overbrace{\begin{bmatrix}\theta_C\\ CS_i, C_i\\ CL_l\\ CH_l\\ \beta\end{bmatrix}}}{\underset{\uparrow}{\begin{bmatrix}ST_s\\ MAS_C, R_{A_C}, R_{BE}\end{bmatrix}}}\rightarrow[PO \quad PE \quad TH] \tag{5.136}$$

We summarize the usage and the simulation context in $EX$, which considers the environmental conditions $\alpha$, the constraints $\eta$, and the application software $SW$. We differentiate the application software by a workload $W$, the utilization time $T_u$, and a set of software-based settings $\xi$, as shown in Equation (5.137).

$$\underset{\overbrace{\begin{bmatrix}\alpha\\ \eta\\ W\\ \xi\\ T_u\end{bmatrix}}}{\underset{\uparrow}{[\,EX]}}\underset{\overbrace{\begin{bmatrix}\theta_C\\ CS_i, C_i\\ CL_l\\ CH_l\\ \beta\end{bmatrix}}}{\underset{\uparrow}{\begin{bmatrix}ST_s\\ MAS_C, R_{A_C}, R_{BE}\end{bmatrix}}}\rightarrow[PO \quad PE \quad TH] \tag{5.137}$$

Table 31 lists the symbols and definitions of the server system externals section.

Table 31: Nomenclature – server system externals (*EX*)

| Nomenclature | Meaning |
|---|---|
| $\boldsymbol{A}$ | Aspect |
| $\boldsymbol{C}$ | Component |
| $\boldsymbol{CS}$ | System-board category $i$ ( $\equiv$ components) |
| $\boldsymbol{EX}$ | Externals |
| $\boldsymbol{SY}$ | System characterization, model |
| $\boldsymbol{FS}$ | Full-system simulation and optimization |
| $\boldsymbol{M_{SY}^{EX}}$ | Mapping between externals and system |
| $\boldsymbol{SW}$ | Application software |
| $\boldsymbol{t_k}$ | Time step $k$ |
| $\boldsymbol{T_F}$ | Time amplifier for workload |
| $\boldsymbol{T_{case}}$ | Processor case temperature |

| Nomenclature | Meaning |
|---|---|
| $\boldsymbol{i, j, n, m, k, l}$ | Index |
| $\boldsymbol{N_0}$ | Any natural number $N_0 = \{0,1,2,3,\dots\}$ |
| $\mathbb{R}$ | Any positive real number |
| $\boldsymbol{\alpha, \hat{\alpha}}$ | Environment conditions |
| $\boldsymbol{\eta}$ | External constraints |
| $\boldsymbol{\xi}$ | Software settings |
| $\boldsymbol{\vec{u}_m = u_{CS_i}}$ $\boldsymbol{\vec{u}_{t_k}}$ | Utilization level of component category $m$ time step $t_k$ |
| $\boldsymbol{w_{il}}$ $\boldsymbol{W, W_{T_u}}$ | Workload for component $m$ at a time $t_k$ |
| $\boldsymbol{WP_k}$ | Workload profile $k$ |
| $\boldsymbol{T}$ $\boldsymbol{T_s, T_u, T_w}$ | Time period: simulation, utilization, workload |
| $\boldsymbol{P_m, P_i, P_c, P_u}$ | Performance counters of memory bus transactions, instructions, clock cycles, last-level cache references |
| $\boldsymbol{K_0, K_1, K_2, K_3, K_4, K_5, K_6}$ | Coefficients |

The external domain does not consider the energy supply system itself, because the server location sets the supply voltage $\alpha_{volt}$[253]. We address the environmental conditions $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_k\}$. The customer cannot change the thermal conditions[254] $\alpha_{temp}$, but can limit the exhaust system temperature[255]. Therefore, our concept considers exterior constraints $\eta = \{\eta_1, \eta_2, \ldots, \eta_k\}$, which are parameters of the management and optimization strategies. Another major aspect of the external domain is the application software $(SW)$ which the server executes. The workload $W$ stimulates the system, whereby we consider the software-related settings $\xi$ and the utilization time $T_u$. The external domain is a set of the software characteristics, the environmental characteristics, and the exterior constraints, see Equation (5.139) and Figure 62.

$$\alpha = \{\alpha_{volt}, \alpha_{temp}, \ldots, \alpha_{others}\} = \{\alpha_1, \alpha_2, \ldots, \alpha_k\}, \alpha_l \in \alpha \tag{5.138}$$

$$EX = \{\alpha, \eta, SW\},\ SW = \{W, \xi, T_u\} \rightarrow\ EX = \{\alpha, \eta, W, \xi, T_u\} \tag{5.139}$$

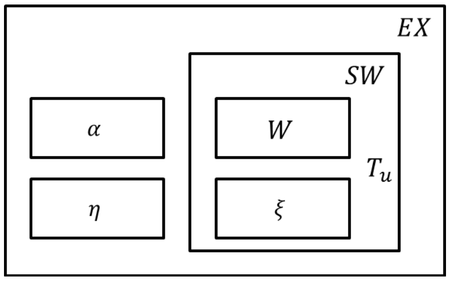


Figure 62: Externals

State-of-the-art, full-system approaches [Bel 2000, BJ 2003, FWB 2007, CBB et al. 2010, KJC et al. 2014] do not consider customer changes or disturbances during the simulation. In addition, state-of-the-art, full-system power models have in common that these approaches are not aware of external constraints, such as thermal conditions or limits provided by the customer.

## 5.3.1 Environment Characterization and External Constraints $\{\alpha, \eta\}$

The approaches in [Bel 2000, JM 2001, MPL 2009] trace application software to optimize the system under specific conditions, but do not concentrate on the physical environment or their external influences. In [BKW et al. 2003, WB 2004, MB 2006], the thermal algorithm considers the ambient temperature as an offset. The algorithm in [RL 2007] uses the system temperature, but does not consider the fan characteristics and thermal development. We consider the external temperature in our approach, which is relevant to the fan power and

[253] Supply voltage: not modifiable, PSU efficiency differs at 220V in comparison to 120V
[254] Thermal conditions: systems inlet / ambient temperature
[255] Exhaust temperature: maximal acceptable temperature because of a limited air-conditioning system (HVAC)

system temperature. The inlet temperature warms the components because the fans suck air from the chassis front inside the system. An input parameter of the simulation model is the ambient temperature $\alpha_{temp}$, which we differentiate into a global system temperature $\alpha_{temp_{SY}}$ and into the local temperatures $\alpha_{temp_{C_i}}$ of each component $C_i$.

$$\alpha_{temp} = \left\{\alpha_{temp_{SY}}, \alpha_{temp_{C_1}}, \alpha_{temp_{C_2}}, \dots, \alpha_{temp_{C_m}}\right\}, \alpha_{temp_{C_i}} \in \alpha \qquad (5.140)$$

The vendor specifies each component by a thermal operating condition, which ensures the full functionality.

$$\eta = \begin{cases} upper\ critical \\ upper\ warning \end{cases} limit \qquad (5.141)$$

If the component reaches an upper limit, the component shuts down to ensure reliability. Otherwise, the component might be damaged as described in Section 3.6. For each component we consider two upper thresholds, defined as a critical limit and a warning limit[256]. Besides the component temperature, the vendor specifies the system temperature.

$$\eta = \left\{\eta_{power}, \eta_{temp}, \dots, \eta_{others}\right\} = \{\eta_1, \eta_2, \dots, \eta_k\}, \eta_l \in \eta \qquad (5.142)$$

$$\eta_{power} = \eta_1 = \begin{cases} upper\ critical \\ upper\ warning \end{cases} limit \qquad (5.143)$$

$$\eta_{temperature} = \eta_2 = \begin{cases} upper\ critical \\ upper\ warning \end{cases} limit \qquad (5.144)$$

$$\eta_k = \left\{\eta_{k_{SY}}, \eta_{k_{C_1}}, \eta_{k_{C_2}}, \dots, \eta_{k_{C_m}}\right\}, \eta_{k_{C_i}} \in \eta \qquad (5.145)$$

A server system usually works in a temperature range from $20°C$ up to $40°C$. We address the upper system temperature as a critical temperature at $40°C$. In [Lin 2009] the authors state that a resource has an upper thermal limit of $+90°C$ to work as designed, otherwise the device error-rate increases significantly. We differentiate in global system limits $\eta_{k_{SY}}$ and local component limits $\eta_{k_{C_i}}$ for each external constraint $\eta_k$. We create a two-dimensional array, which includes the external constraint $\eta_k$ in the first dimension, and the system components $C_i$ in the second dimension. The thermal design power (TDP) commonly specifies the processors' upper critical power limit, whereby the upper warning limit is a range between 80 or 90 percent of the critical power.

$$\eta_{power_{proc}} = \begin{cases} TDP \\ [0.8 * TDP \dots 0.9 * TDP] \end{cases} \qquad (5.146)$$

[256] Limits: A liquid cooling approach requires the lower limits (critical and warning) to ensure working conditions.

The thermal thresholds result from the power consumption, the system airflow, and an adjustment of the component. We assume that the upper critical temperature $max(T_{case})$ is linearly proportional to the ratio between $P_{min}$ and $T_{case}$. The curve gradient $m_g$ depends upon the component material.

$$\eta_{temp_{proc}} = f(m_g, \eta_{power_{proc}}, T_{case}) \tag{5.147}$$

Figure 63 shows the processor's upper critical temperature $max(T_{case})$, which directly derives from the TDP value.

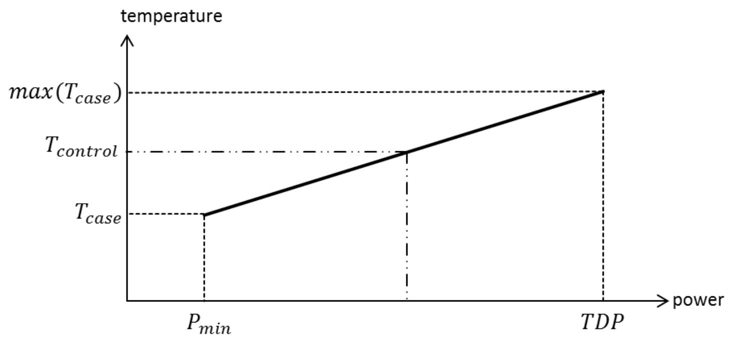


Figure 63: Thermal profile diagram [Int 2014]

Therefore, for each server system the vendor specifies a critical or warning temperature. Equation (5.148) shows the upper thermal and power limits of an *Intel Xeon* processor *E5-2603v2*[257]. The memory vendor defines the commercial temperature in the spreadsheet of each module.

$$\eta_{power_{proc}} = \begin{cases} 80W \\ 70W \end{cases}, \eta_{temp_{proc}} = \begin{cases} 90°C \\ 70°C \end{cases} \tag{5.148}$$

In [BHS 1998], the environment includes the system workload $W$ without any limits. We extend the approach by using the external constraints $\eta$ and the ambient temperature $\alpha$.

### 5.3.2 Software Characterization $\{W, \xi, T_u\}$

Our concept supports various application software considering diverse designs and architectures independent of the hardware configuration. We characterize the software by a workload model, which represents processes or refers to functions. The authors of [BHS 1998] explain that the precise accesses to resources are unknown in the early design stage of the system. A specification at a highly detailed access level results in a too slow simulation. Therefore, the authors define the external environment as requests and the components as resources. We abstract the software variations and their heterogeneous tasks into utilization levels, which are unknown at this abstraction level. We cover high-intensity and low-intensity workload phases in our workload model because the server workload varies over the time

[257] Processor power and thermal limits (profile): defined in a thermal design guideline [Int 2014]

from seconds to days, see Figure 27. Our workload model has to support the software variability on a flexible timescale that considers synthetic workloads, such as benchmarks, to emulate various server usages. More specifically, we follow the concept of [RL 2007], which monitors diverse application software to include the utilization-based traces as simulator stimuli. The synthetic workload provides a simple comparison between the simulation and the real system, which is reproducible at any time [IM 2003, Riv 2008, BGM et al. 2010]. Common full-system simulators [RHW et al. 1995, Her 1998, GSI et al. 2002, MCE et al. 2002, CDS 2003, HSW et al. 2004, RL 2007, Lan 2007] characterize particular software as instruction sets or trace the utilization levels from artificial benchmarks. The authors of [YSY et al. 2011, KJC et al. 2014] describe the workload using specific instructions, operations, or cycle-accurate tracings of various tasks. The instruction-based approach in [BC 2010] is adequate in its accuracy, but the approach depends upon the explicit hardware and its architecture. The authors of [Che 2006] conceive an instruction-based processor model, whereby the memory, cache, and peripherals rely upon the system accesses. We abstract the specific instructions, accesses, and operations by a flexible workload considering diverse utilization levels, which encapsulate the software requests from the hardware architecture to reduce the simulation complexity. As a result, we provide a ratio between the floating-point or integer instructions. We differentiate the processor operations because of their diverse performance results and power consumptions. We specify the workload independent of the operating system, but with regard to the standardized synthetic benchmarks, which utilize the components at various levels. We define workload scenarios to an abstract of an explicit usage, with the premise of the scalability and flexibility.

Moreover, we address the worst-case workload, which the industrial tools of Dell, Fujitsu, HP, and IBM commercial use. In addition, we offer customer-specific use case scenarios, which can be real application traces or an estimated workload behavior. The realistic workloads support the power and over-estimation reduction because of the significant difference between worst-case assumption and authentic behavior. We define weight coefficients for high-intensive, medium-intensive, or low-intensive utilization levels to integrate diverse application software, and their topology, hierarchy, generation, or architecture.

At first, we gather our resources, which rely upon various request types. As a result, we classify the workloads into resource-bounded utilization levels. Secondly, we develop heterogeneous workload scenarios, which cover the use cases of the industrial tool as well as the academic approaches. Herein, we consider the complex system interactions, which result in any mutual influence of resource utilization. In the next step, we consider the software-based setting, which depends upon the explicit workload or describes influences that do not rely upon the software execution.

#### 5.3.2.1 Resource Clustering

The purpose of the system can change during the life cycle. In a virtualized environment, the data center manager schedules the services and tasks just in time, finally scheduling the requests in any manner, because of fluctuating customer demands. We cannot predict the data center scheduling or resource planning techniques in an early design stage. Data center resource planning tools autonomously shift virtual machines from one server to another. For that reason, state-of-the-art, full-system simulators decouple the workload from the physical domain using utilization levels instead of instructions or transactions. The vendors characterize the system independently of how and for what purpose the system is used.

The abstract utilization-based scenarios in [Riv 2008, RRK 2008] consider the explicit hardware, whereby the software profiles are based upon the performance counters $P_m,\ P_i,\ P_c,\ P_u$, which correspond to the number of memory bus transactions, the number of instructions retired, the unhalted clock cycles, and the number of last-level cache references. The authors use the coefficients $K_i$ to quantify the processor or disk utilization, including the memory-related instructions and transitions. The resource-specific addition or subtraction results from the exact workload scenario. A processor-bounded workload includes less disk utilization in comparison to a memory-bounded workload.

$$P = K_0 \mp K_1 * \left\{\frac{u_{proc}}{\max(u_{proc})}\right\}^1 \mp K_2 * \frac{u_{disk}}{\max(u_{disk})}$$

$$\mp K_3 * \frac{P_m}{\max(P_m)} \mp K_4 * \frac{P_i}{\max(P_i)} \mp K_5 * \frac{P_c}{\max(P_c)} \mp K_6 * \frac{P_u}{\max(P_u)} \tag{5.149}$$

Performance counters are specific to the architecture, generation, and family. Therefore, we avoid them to conceive a generic model. The full-system simulation in [RL 2007] uses resource-bounded utilization levels without performance counters. The authors concentrate upon processor-bounded workloads, which is state-of-the-art in the industrial tools. The authors of [GFN et al. 2006] define the common resources as a set of components $C$.

$$C = \{C_1, C_2, \dots, C_m\},\ C_i\ \in C \tag{5.150}$$

The authors of [ERK 2006] utilize especially the resources processor, memory, disk, and network, as shown in Table 32. The authors of [ERK 2006, Riv 2008] define resource-bounded utilization levels.

Table 32: Definition $C_i$

| Terms $C_i$ | Component |
|---|---|
| $C_1 = proc$ | $processor$ |
| $C_2 = mem$ | $memory$ |
| $C_3 = dis$ | $disk$ |
| $C_4 = net$ | $network$ |

In [GFN et al. 2006], every component is a set of sub-components. In general, we group the resources into the categories $CS$ to support flexible utilization types. Each category $CS_i$ consists of a set of components $C_i \in C$ to cover heterogeneous application software or benchmark behavior, which utilizes various components.

$$CS = \{CS_1, CS_2, \dots, CS_m\}, CS_i \in CS \tag{5.151}$$

$$CS_i = \{C_1, C_2, \dots, C_m\}, C_i \in CS_i \tag{5.152}$$

We restrict ourselves to the categories *processor*, *memory*, *input/output*, *fan*, and *others* ($CS = \{proc, mem, io, fan, oth\}$) to cover category-bounded workload types. The industrial tools address processor-bounded workloads. The authors of [RRK 2008] differentiate into processor-intensive and non-processor-intensive workloads. Inside [YP 2009], the authors distinguish into computational-bounded and memory-bounded workloads. The authors of [BC 2010] classify the workloads into processor-bounded and input/output-bounded processes. We differentiate non processor-intensive workloads into memory-bounded and I/O-bounded ones to cover a realistic workload and have a workload suitable for synthetic benchmarks.

Table 33: Definition $CS_i$

| Terms $CS_i$ | Categories |
|---|---|
| $\boldsymbol{CS_1 = proc}$ | $processor$ |
| $\boldsymbol{CS_2 = mem}$ | $memory$ |
| $\boldsymbol{CS_3 = io}$ | $input/output$ |
| $\boldsymbol{CS_4 = fan}$ | $fan$ |
| $\boldsymbol{CS_5 = oth}$ | $other$ |

If we consider the approach of [ERK 2006], then the category $CS_3$ considers the components $C_3 = disk$ and $C_4 = network$.

$$CS_3 = \{C_3, C_4\} = \{dis, net\} \tag{5.153}$$

In our concept, we simplify the category into the related components $CS_1 = C_1$ and $CS_2 = C_2$ for our exact server configuration. We do not cover the sub-components of $CS_3$, because we abstract from the network communication to reduce the complexity. We consider the utilization of $CS_4$ components in the thermal model. The category $CS_5$ primarily covers the mounted system-board components, which build the base power. We categorize our utilization levels for each defined category $CS_i \in CS$, whereby we do not cover the fans.

$$u_{CS} = \{u_{proc}, u_{mem}, u_{io}, u_{oth}\} = \{u_{CS_1}, u_{CS_2}, \dots, u_{CS_m}\} \tag{5.154}$$

As a restriction, our utilization-based approach is not able to cover memory allocation techniques or access patterns on the physical domain, as shown in [YVK et al. 2000, Bel 2001, SBM 2009, HCE et al. 2011, RAK et al. 2013, LSQ et al. 2014].

### 5.3.2.2 Workload Scenarios and Profiles $\{W, T_u\}$

We characterize application software concerning the power, thermal, and performance characteristics. We use the category-specific utilization levels $u_{CS}$ to create a workload $W$, which stimulates the simulation model. A workload scenario consists of diverse profiles that consider various utilization levels. We enhance the approach of [ERK 2006, BJ 2007, Riv 2008] considering customer-specific scenarios in our profiles. The workload is configurable by the customer on the basis of the category-specific utilization levels at a time resolution of a minimum of one second. Our aim is to achieve an average, minimum, or maximum utilization of the components to create a realistic image of application software. We can balance the utilization levels to present the workload variations. The same approach is suitable to emulate synthetic benchmarks.

The industrial tools concentrate upon processor-bounded $u_{CS} = \{u_{proc}\}$ workloads. Academic approaches use additionally memory-bounded $u_{CS} = \{u_{mem}\}$ workloads because of the synthetic benchmark-based evaluation. We do not concentrate upon I/O-bounded workload $u_{CS} = \{u_{io}\}$ as done in [BC 2010], but consider the corresponding utilization levels in our workload definition. We consider the processor operation type (floating-point, integer) and memory access type (read, write) but do not involve the exact instructions. We assume category-specific details in the software settings $\xi$. We generalize the concept of [Riv 2008], the flexible configuration of categories, and related utilization levels, which we map to the specific components in dependence on their characteristics. We merge the diverse category-bounded utilization types to consider and reflect an image of the customer-specific workload scenarios. We cover a steady workload to reproduce the industrial benchmarks and consider changing workloads to support academic approaches.

According to [BHS 1998], we characterize the application software as a series of external requests over time. We use a time resolution of $\Delta T = t_2 - t_1 \geq 1s$ between two utilization levels because a smaller time scale creates more computational effort and results in a long simulation time. The processor changes the frequency within a microsecond or millisecond, for instance. We have to calculate the aspect $A_j$ at every time step $t_k$, which increases the computation time. The system sensors, especially the temperature sensors, work within seconds because of the bus latency and bandwidth. In addition, the component and thermal inertia guide at a time schedule bigger than one second does. Our approach does not cover a workload in a 24-hour or weekly time base, because of the resulting storage demands. We assume a workload of a maximum of one hour. We specify for each point in time $t_l$ the category-specific utilization levels $u_{CS_i}$ in $\vec{U}_{t_k}$. We conclude the utilization levels $\vec{U}_{t_k}$ for a time $T_u = \sum_0^l t_k$ to define a time-annotated workload scenario $W_{T_u}$. We generalize the workload $W_{T_u}$ into a two-dimensional array $W$, whereby the first dimension is the time $T_u$ and the second dimension is the specific category-based utilization levels.

$$\vec{U}_{t_k} = \{u_{CS_1}, u_{CS_2}, \dots, u_{CS_m}\} \text{ at } t_k, t_k \in T_u \tag{5.155}$$

$$W_{T_u} = (\vec{U}_{t_1}, \vec{U}_{t_2}, \cdots, \vec{U}_{t_k}) \tag{5.156}$$

$$W = (w_{il}) = \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1k} \\ u_{21} & u_{22} & \cdots & u_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ u_{m1} & u_{m2} & \cdots & u_{mk} \end{pmatrix}; \; i, m \in CS; \; l, k \in T_u \tag{5.157}$$

$$W = (w_{il}) = \begin{pmatrix} u_{proc_{t_1}} & u_{proc_{t_2}} & \dots & u_{proc_{t_k}} \\ u_{mem_{t_1}} & u_{mem_{t_2}} & \cdots & u_{mem_{t_k}} \\ u_{io_{t_1}} & u_{io_{t_2}} & \cdots & u_{io_{t_k}} \\ u_{oth_{t_1}} & u_{oth_{t_2}} & \cdots & u_{oth_{t_k}} \end{pmatrix}; \; i, m \in CS; \; l, k \in T_u \tag{5.158}$$

A scenario $W_{T_u}$ can include multiple workload profiles $WP = \{WP_{T_1}, WP_{T_2}, \dots, WP_{T_k}\}$, whereby $T_u = \sum_k^u T_k$. We divide a time $T_i$ into $l$ steps $T_i = \sum_0^l t_l$.

$$WP_{T_1} = \begin{pmatrix} u_{10} & u_{1(k/l)} & \dots & u_{11} \\ u_{20} & u_{2(k/l)} & \cdots & u_{21} \\ \vdots & \vdots & \ddots & \vdots \\ u_{m0} & u_{m(k/l)} & \cdots & u_{m1} \end{pmatrix}; \; m \in CS; \;\; k, l \in T_u \tag{5.159}$$

We define full utilization of a component by $u_{CS} = 100\%$ and define the component idle state as the minimal utilization level $u_{CS} = 0\%$. The utilization levels define the component usage in relation to its maximal available physical working capacity $\frac{u_{CS_i}}{\max(u_{CS_i})}$. A utilization level is the percentage of time the component spends doing work in contrast to being idle. The workload can completely utilize single or multiple components up to 100%. A full-system utilization has at least one utilization level $u_{CS_i}$ at $100\%$, which we define as a $CS_i$-bounded workload. We support various profiles, such as processor-bounded workloads, which are configurable by the customer and related to the category-specific profiles.

The customer specifies the workload profile $WP_{T_k}$ at a time $T_w$ of any $u_{CS_i}$. Our scenario might have a longer utilization time $T_u$ in comparison to $T_w$. We assume that the utilization values $\vec{U}_{T_u}$ are constant in the interval between $T_w$ up to $T_u$. Moreover, our scenario $W_{T_u}$ might consist of various benchmarks, which have a smaller time period $T_u$ in comparison to the required simulated time $T_s$. It is vital that we close the gap between $T_s$ and $T_u$. We suggest two utilization-based approaches; at first, we reuse the last utilization values $\vec{U}_{T_u}$ and consider steady values up to $T_s$. Second, the utilization values from the first-time interval recursively fill the gap. Alternatively, the customer can define and amplify each utilization time step $t_k$ using a time factor $T_F$ to reduce the gap between $T_u$ and $T_s$. We equidistantly expand the time of the previous utilization levels.

We conceive a time-based workload that generates the simulation stimuli considering various category-specific utilization levels of diverse application software or benchmarks. Our *utilization generator*, see Figure 64, supports steady and continuous workload scenarios. We expand the industrial steady-state scenarios by extending novel dynamic profiles, whereby we flexibly annotate timing for successive scenarios. We emulate a real customer-specific workload, which we simulate for the server configuration.

In contrast, our workload does not consider the fine-grained transactions of each component. Our concept neglects the impact of diverse batch sizes, transaction mixes, threads, or queue categories[258], which the authors in [KGS 2008] observe. We propose a flexible workload model considering the category-specific utilization levels, which we restrict to cover the real behavior of diverse benchmarks or application software.

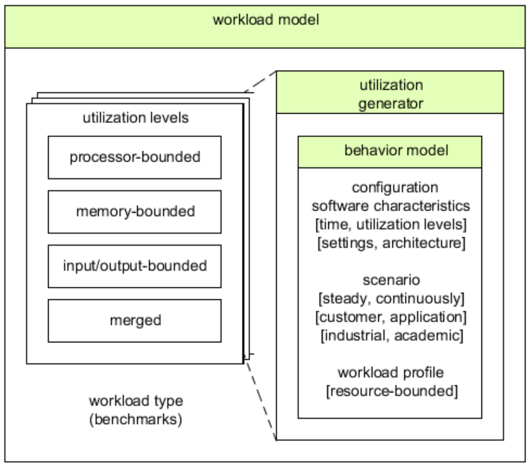


Figure 64: Workload model contribution

#### 5.3.2.3 *Software-based Influences and Settings ξ*

In the early design stages, we do not know about the implicit read or write accesses of a memory-intensive workload. We cannot predict the processor floating-point or integer instructions for novel systems.

The authors in [Fuj 2011] state that using a Windows operating system (OS) consumes approximately twenty percent less than the same system using a Linux OS when executing the identical *SPECpower* benchmark. Both OS are not comparable, because of their architecture or background threads. The Microsoft Windows OS provides a complete ACPI support, which enables all processor states. In contrast, some Linux OS versions cannot control the processor in the same detailed manner. The customer chooses any operating system[259] for the server system. In the work of [RHW et al. 1995, Her 1998, Lan 2007] the models are exact OS images

[258] Queue categories: single vs. multiple

[259] Operating systems: popular server OS are Red Hat Enterprise Linux (RHEL), SUSE Linux Enterprise Server (SLES), and Windows Server, for instance

including explicit operations to support the resource optimization. The authors of [TRJ 2005] analyze the OS to optimize the power consumption on the basis of executed instructions and interactions. We cannot apply these approaches for a generic model, as each OS and system generation handles the tasks differently because of the system architecture. The complexity, the amount of possible OS, and the measurements lead to extra efforts. Therefore, we abstract the OS and exclude precise OS models, which require a millisecond resolution. We decide to include the OS type as a global external simulation setting $\xi_1$, which is independent of the internal architecture and instructions. We do not further investigate on the explicit OS-dependent power consumption, but consider the fact obtained from the empirical analyses of Fujitsu [Fuj 2011] that a Microsoft Windows OS consumes less power than a similar Linux operating system because of the extensive ACPI support that highly efficient handle the different power states. We consider an OS-specific correction factor in our total power calculation, which the customer specifies as an empirical and static simulation parameter.

The BIOS/UEFI settings influence the energy efficiency of the server system [KGS 2008, Fuji 2010]. The authors in [EM 2010] present that a *Turbo Mode* "…, will evaluate the utilization of the CPU and will not engage unless additional performance has been requested by the OS for a period of 2 seconds." This power optimized processor setting is significant for workloads with a dynamic frequency scaling. The customer can only change the BIOS/UEFI settings when the server system is off. For instance, the customer configures the virtualization support, which enables additional processor states. As stated in Section 3, academic and industrial approaches do not cover such details, because the authors assume and analyze fixed settings. We consider the hardware-based BIOS/UEFI settings $\xi_2$ as a software configuration, which enables a rapid assessment of the energy efficiency effect. Alternatively, we can use the settings as a static input simulation parameter, which needs multiple simulation runs. The same demand is observable for the firmware settings. The customer can change the power and thermal limits $\xi_{l_{C_i}}$ in the firmware, which directly influences the power management techniques. We characterize the software-based settings as a two-dimensional array, whereby the first dimension specifies the kind $l$ and the second dimension defines the related component $C_i$.

$$\xi = \left\{\xi_{1_{C_1}}, \xi_{2_{C_2}}, \dots, \xi_{k_{C_m}}\right\}, \xi_{l_{C_i}} \in \xi \tag{5.160}$$

$$\xi_1 = \xi_{OS},\ \ \xi_2 = \xi_{BIOS/UEFI},\ \ \xi_3 = \xi_{FW},\ \ \xi_4 = \xi_{SW} \tag{5.161}$$

Figure 65 shows the environmental conditions, external constraints, and the software, which includes the workload model and their corresponding settings.

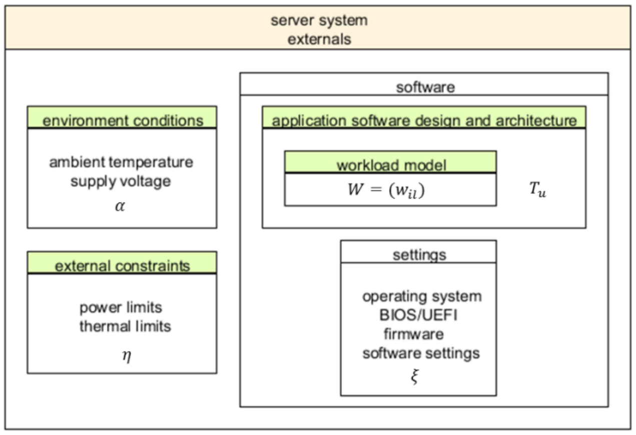


Figure 65: Server system externals

We encapsulate the externals from the explicit system configuration and use the software application space, defined as workload $W$, to stimulate the server system considering the software settings $\xi$, the environmental conditions $\alpha$, and the external constraints $\eta$. It altogether builds the externals $EX$, see Equation (5.162). We map the externals to the concrete system platform by $M_{SY}^{EX}$ in the *process and control* layer. We concretely scale the utilization levels to the system components that rely upon their quantity and characteristics. For instance, the customer specifies a processor utilization level at 100 percent. We configure the 100 percent for all processors within the system. We have to adapt the utilization level when we change the processor characteristics or disable a processor. The authors of [SM 2001] map the software to the hardware platform using the network communication. The externals, especially the application software, define the ordered sequence of actions which the simulation has to perform considering the constraints $\alpha$ and $\eta$.

$$EX = \{\alpha, \eta, W, \xi, T_u\} \tag{5.162}$$

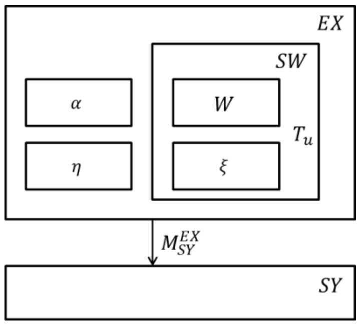


Figure 66: Externals and system

The *externals* define the environmental conditions, external simulation constraints, customization parameter of the simulation model, and the customer-specific workload scenario.

## 5.4 Server System Optimization $FS = \{EX, SY\}$

We create the simulation model on the basis of the operational models in Section 5.2 to find a set of solutions (Pareto front) across the flexible workload scenarios, see Section 5.3. The simulation run considers a customer-specific, exact workload scenario, and a server system configuration. We do not dedicate the simulation to an explicit server system or particular operating system, as done in [RHW et al. 1995, GSI et al. 2002, HSW et al. 2004]. Our simulation framework supports the variance of server systems and generations on the basis of a flexible configuration and characterization. Our aim is to find at least one optimal solution concerning an energy efficiency ratio. We consider a set of parameters – decision variables – forming the design space of our *optimization strategy*. The nomenclature in Table 34 and Table 35 briefly list the symbols of this section.

Table 34: Nomenclature – server system simulation $(I)$

| Nomenclature | Meaning |
|---|---|
| $A$ | Aspect |
| $C$ | Component |
| $CS$ | System-board category ( $\equiv$ components) |
| $CL$ | Classes |
| $SY$ | System |
| $FS$ | Full-system simulation and optimization |
| $AC$ | Architecture |
| $CC$ | Connectors |
| $EE, EE_{BASE}$ | Energy efficiency |
| $PO$ | Power |
| $PE$ | Performance |
| $TH$ | Thermal |
| $proc$ | Processor |
| $mem$ | Memory |

| Nomenclature | Meaning |
|---|---|
| $i, j, n, m, k, l$ | Index |
| $N_0$ | Any natural number $N_0 = \{0,1,2,3, \dots\}$ |
| $\alpha, \hat{\alpha}$ | Environment conditions |
| $\delta, \hat{\delta}$ | Communication |
| $\gamma, \hat{\gamma}$ | Management techniques |
| $\xi$ | Software-based settings |
| $\varepsilon$ | Simulation results |
| $\eta$ | External constraints |
| $\upsilon$ | Internal constraints |
| $\chi$ | Simulation constraints |
| $\sigma$ | Abort criterion |
| $x, \check{x}$ | Decision variable, modified |
| $L_1$ | First level of a certain tree |
| $\theta$<br>$\theta_R, \theta_C, \theta_S$<br>$\theta_{TS}, \theta_{CS}$ | Configuration tree (HW,SW)<br>released, customer-specific, system-compatible<br>technical specification, characteristics |

Table 35: Nomenclature – server system simulation $(II)$

| Nomenclature | Meaning |
|---|---|
| $io$ | Input/output |
| $oth$ | Others |
| $MAS_C$ | Aspect-based models per component |
| $A_{jC_i}$ | Element in matrix $MAS_C$ |
| $F_{A_{jC_i}}$ | Functional description of $A_{jC_i}$ |
| $CH, CH_{TS}$ $CH_{CFG}^{ST}, CH_{CFG}^{DY}$ | Characteristics: technical specification, static, dynamic configuration |
| $WF_{A_{jC_i}}^{CH}$, $WF_{A_{jC_i}}^{CL_{TS}}$, $WF_{CFG}^{ST}$, $WF_{CFG}^{DY}$ | Weight coefficients: component $i$, aspect $j$, and their characteristics, class, static, dynamic |

| Nomenclature | Meaning |
|---|---|
| $\theta_C^l$, ${}_k\theta_C^l$ $\theta_C^l(x)$ | Customer-specific configuration at iteration $l$, time step $k$, and decision variable $x$ |
| $f_{min}, f_{max}$ | Minimal and maximal frequency |
| $\vec{u}_m = u_{CS_i}$ $\vec{u}_{t_k}$ | Utilization level of component category $m$ time step $t_k$ |
| $w_{il}$ $W, W_{T_u}$ | Workload for component $i$ at a time $t_k$ |
| $t_k$ | Time steps $k$ |
| $T$ $T_s, T_u, T_w$ | Time period (timespan) simulation, utilization, workload |
| | |

The constituents of our simulation model are *simulation*, *results*, and *optimization strategy*, see Figure 67. *Simulation* includes the computational cycles to specify the energy efficiency ratio under various conditions, whereby the set of decision variables result from the *optimization strategy* block. We analyze the energy efficiency, decide about the management techniques, and alter the configuration or characteristics as part of our *optimization strategy*. We observe the power, temperature, and performance of each component as well as of the entire server system. We trace the actual decision variables, category-specific utilization levels, and the energy efficiency ratio, which the *results* block presents graphically. The *externals* specify the environmental working conditions of the server system. We consider all simulation inputs and parameters as constant during the simulation process, e.g., the environmental conditions $\alpha$, external constraints $\eta$, and software settings $\xi$. The simulation constraints $\chi$ define the thermal limits or the maximal utilization level of the components at the same time.

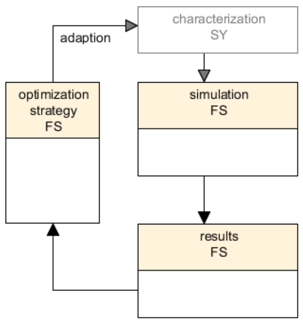


Figure 67: Server system simulation constituents

We concentrate upon an individual workload scenario and an explicit customer-specific server configuration $\theta_C$ during a simulation run. In our prototype implementation, we limit the hardware configuration towards rack-based and tower-based server systems because we want to reduce the degree of freedom of the simulation model by specifying the actual system architecture. We have to consider the specific slots to mount several devices, which might share the input-output devices, or especially the power supply units when we support blade servers, see Section 2.4.1. We predefine the server system architecture $AC$, communication $\delta$, and connectors $CC$ in our simulation manually. Our operational models are based on sensor[260] results gained from the empirical analysis of the real system. The latency and bandwidth of the sensors restrict the minimal sample time, which we specify on a 1-second basis. In our prototype implementation, we restrict the time steps $(t_{k+1} - t_k)$ to last at least one second because we do not consider a cycle accurate or instruction set simulation that would require a higher resolution [Her 1998, MCE et al. 2002, CDS 2003, CDS 2007]. In our simulation framework, a smaller sample time does not provide more details when we specify time-discrete and value-discrete stimuli. The sample time should be adequate to ensure the balance between the computation time, data size (sample time), and accuracy. If the customer specifies the time as hours, we adjust our sample time considering the Nyquist-Shannon theorem, see Section 3.2. The workload scenario $W_{T_u}$ builds the continuous stimuli of our simulation, which reproduces the OS-independent, realistic observations, or assumptions of the customer-specific applications. The customer specifies for each point of time $t_l$ the category-specific utilization levels $u_{CS_i}$ in $\vec{U}_{t_k}$ for a timespan $T_u = \sum_0^l t_k$, which has $(l-1)$ discrete steps. The time-based vector of the utilization levels $\vec{U}_{t_k}$ provides discrete values, which we separately consider at each time step $t_k \in T_u$. We optimize the server system over the entire utilization time $T_u$, which is a long-term[261] aim.

[260] Sensors: power and temperature sensors
[261] Long-term: time horizon of hours, days, weeks, or years

In the pre-process of the server system optimization we define the set of decision variables on the basis of the initial configuration $\theta_C$, considering the workload conditions. We restrict the set of configurations and characteristics, which may save energy in comparison to the original configuration. We initialize the aspect-based calculation methods in $MAS_C$ and specify the base energy efficiency $EE_{BASE}$ assuming $\theta_C$. We prepare the calculation methods considering the flexible range of values gained from the set of decision variables, which we update during the optimization. After a step-based analysis of the energy efficiency ratios, we iteratively decide on the alternation of further decision variables relying upon the best admissible local solution and their potential improvement. We store the energy efficiency ratios in a descending list and update them at each iteration, which is a rudimentary part of the optimization strategy. In accordance with an alternation $\theta_C^l(x)$, we modify the calculation methods[262], estimate the energy efficiency $EE(\theta_C^l, \vec{U}_{t_k})$, and analyze the updated values on the basis of the previous iteration.

The energy efficiency ratios and their related configurations are a sequence of local solutions because we optimize the server system at each point in time specified as short-term[263]. In principle, we finish our optimization process at a specific time step and advance the time whenever we have considered all possible alternatives or combinations of the decision variables at this time step. We apply a heuristic to speed up the optimization strategy. At the end of our simulation, we consider the ranked lists in descending order sorted by their energy efficiency ratios. We analyze the local solutions stored in the lists, considering all time steps in $T_u$ to find a long-term global optimum or a set of long-term optimal solutions. In the global analysis, we study what solution dominates over the entire simulated time. Figure 68 and Figure 69 show the workflow of the optimization process and simulation, which we outline in the next sections.

[262] Modification of calculation methods: change configuration-specific and characteristic-specific coefficients and values

[263] Short-term: time horizon within microseconds, milliseconds, seconds, or minutes

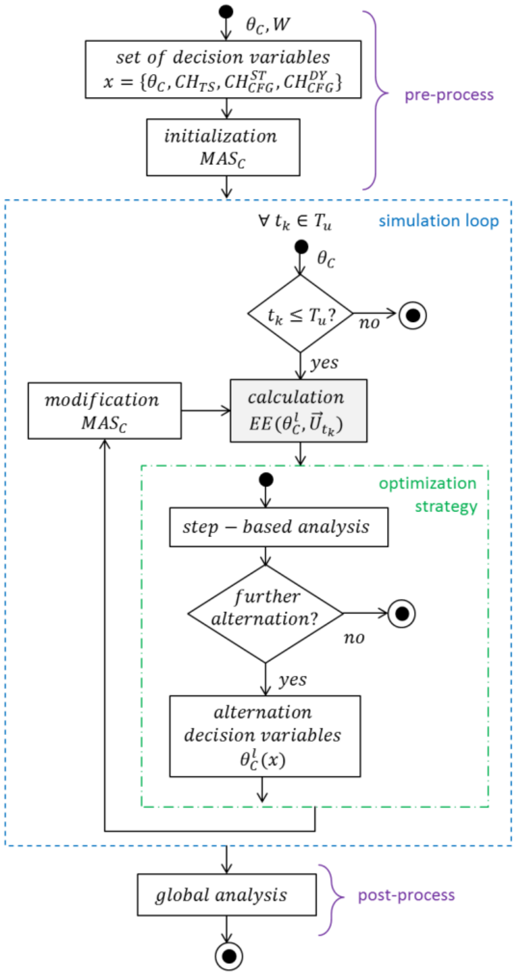


Figure 68: Workflow of the optimization process and simulation

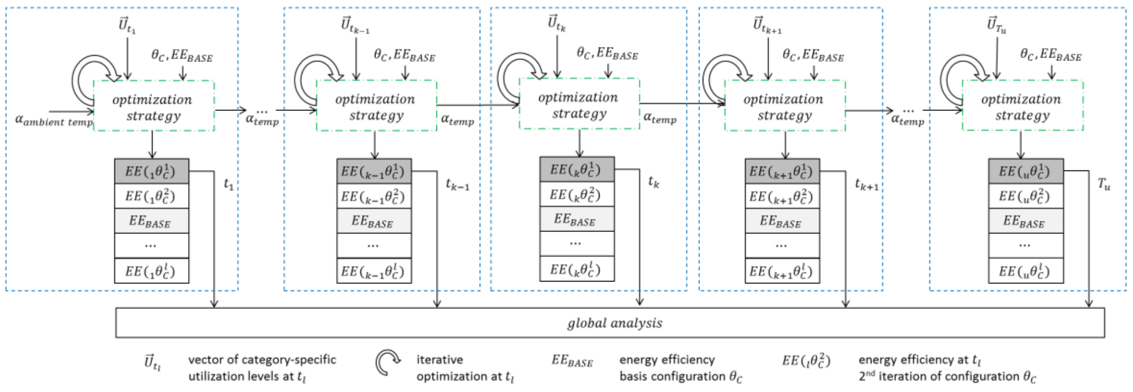


Figure 69: Graphical presentation of the optimization process and simulation

### 5.4.1 Pre-Process

Before the simulation starts, in the simulation pre-process we define the set of the decision variables that form the design space of the *optimization strategy*. In the first pre-process step, we restrict the generic configuration tree $\theta$ to reduce the design space. The tree contains the customer-specific configuration $\theta_C$, which is a released configuration, especially for the customer order process. In the design phase, we concentrate upon the system-compatible configurations.

We consider the technical specification in $\theta_{TS}$ followed by the characteristics in $\theta_{CS}$, which we distinguish among the classes and characteristics. We search the components $C_i$ of $\theta_C$ by a breadth-first search (BFS) algorithm at the level $L_1$ of the generic configuration tree, which builds the root type of the technical specification tree. We consider at least one path of each category in the generic configuration tree. We split up the trees regarding the categories for parallel processing in our simulation model. We skip to the next level of $\theta$ and search all possible characteristics and configurations of $\theta_C$. We distinguish into the actual settings and their alternatives. We reversely start a depth-first search (DFS) algorithm beginning in the leaves to find the system-compatible hardware and copy the tree, wherein we remove the incompatible configurations by their subtypes, as shown in Figure 70. In this example, the actual memory configuration is $SDRAM\ DIMM, DDR3, DDR3-1600, PC3-12800$, which exemplarily has the alternative configurations $DDR3-1333, PC3-10600$ and $DDR3-1866, PC3-14900$. We always consider system-compatible hardware only, which we can release to the customer. The server system specification restricts the hardware, which we can assemble into the enclosure. The limitations in the technical specification tree reduce the simulation time because of reduced complexity.

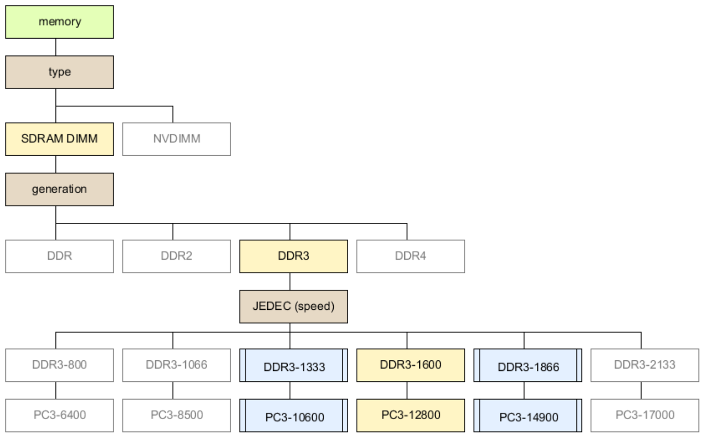


Figure 70: System-compatible technical specification tree

Our aim in the next phase is to restrict the possible range of characteristics in $\theta_{CS}$, beyond which we cannot adjust by our optimization strategy. An example is the frequency range $[f_{min}, f_{max}]$, which we cannot exceed during our optimization. We start the same techniques in accordance to the technical specification tree and set the actual leaves as the root type of each characteristic tree $\theta_{CS}$. Furthermore, we differentiate between the technical, static, and dynamic characteristics, which refer to the respective classes. We sort the classes $CL$ by the corresponding tree, including the associated characteristics. Finally, we generate a copy of $\theta$ considering the restrictions on the basis of the customer-specific configuration, as exemplarily shown in Figure 71.

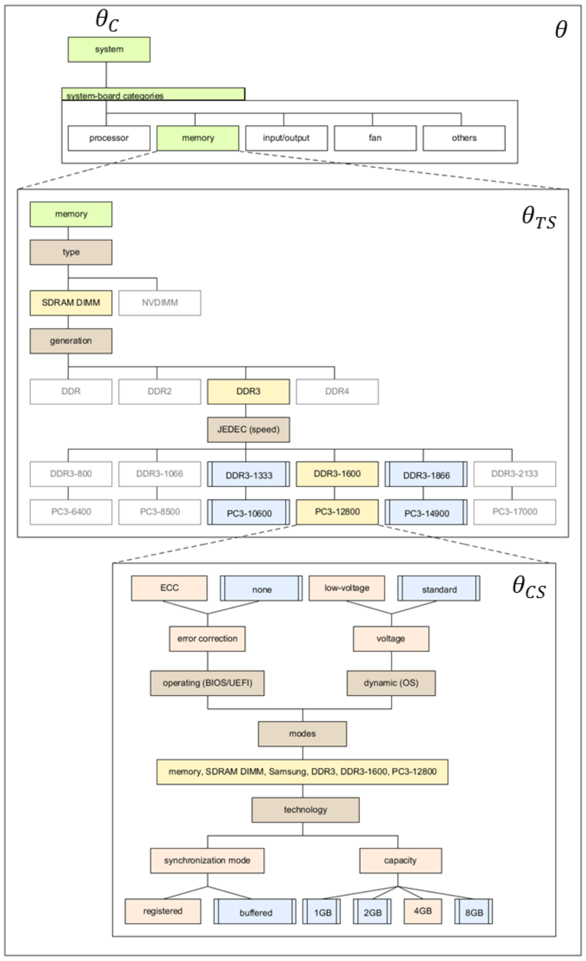


Figure 71: Memory tree

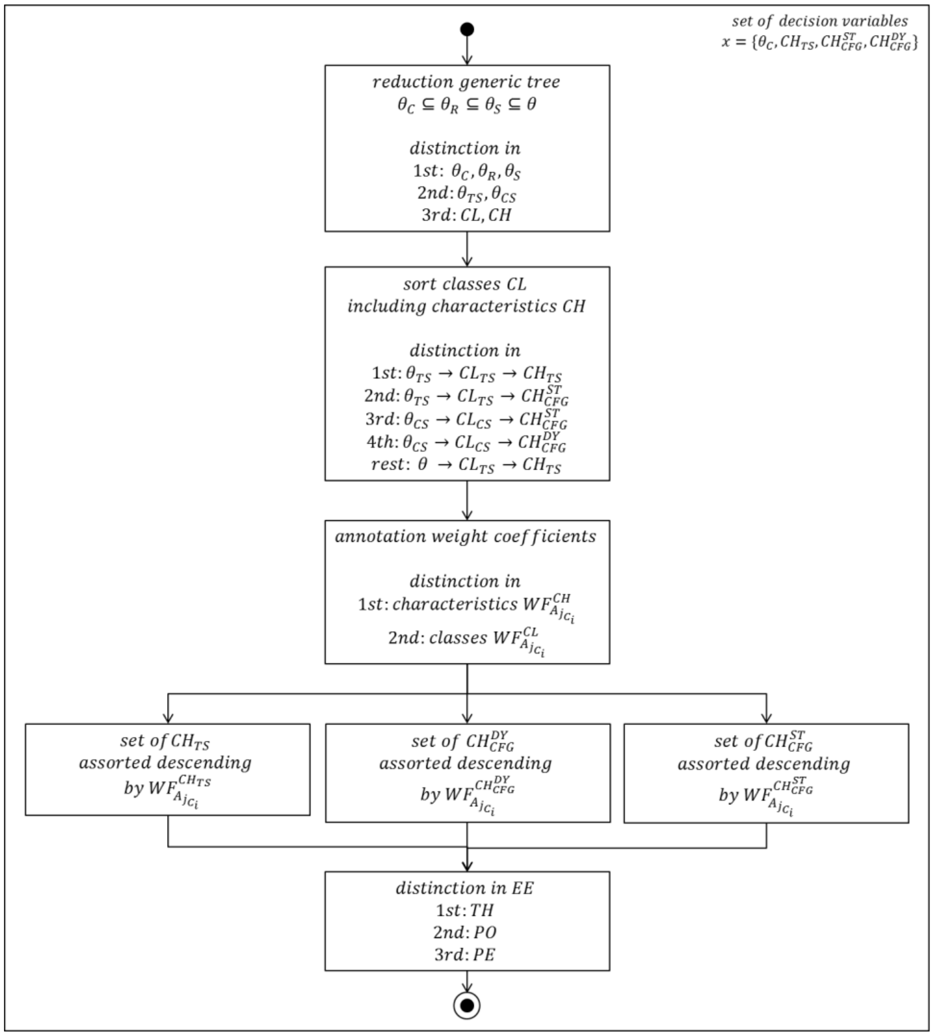
As part of the tree generations $\theta_{TS}$ and $\theta_{CS}$, we annotate the weight coefficients $WF_{A_{j_{C_i}}}^{CH}$ of the characteristics which we require in the utilization-based functions $F_{A_{j_{C_i}}}$ within the matrix $MAS_C$. We sum up all weight coefficients of one subdivision $\{CH_{TS}, CH_{CFG}^{ST}, CH_{CFG}^{DY}\}$ into the class-specific weight coefficients.

$$\sum WF_{A_{jC_i}}^{CH_{TS}} = WF_{A_{jC_i}}^{CL_{TS}} = WF_{TS} \tag{5.163}$$

$$\sum WF_{A_{jC_i}}^{CH_{CFG}^{ST}} = WF_{A_{jC_i}}^{CL_{ST}} = WF_{CFG}^{ST} \tag{5.164}$$

$$\sum WF_{A_{jC_i}}^{CH_{CFG}^{DY}} = WF_{A_{jC_i}}^{CL_{DY}} = WF_{CFG}^{DY} \tag{5.165}$$

We generate a set of each subdivision in descending order of their weight coefficients and distinguish between the diverse aspects in the energy efficiency. We analyze the characteristic and classes that have the most impact upon the energy efficiency ratio and define the execution sequence of the optimization strategy. Figure 72 graphically presents the first step of the pre-process, which defines the process to set the decision variables.



set of decision variables
$x = \{\theta_C, CH_{TS}, CH_{CFG}^{ST}, CH_{CFG}^{DY}\}$
reduction generic tree
$\theta_C \subseteq \theta_R \subseteq \theta_S \subseteq \theta$
distinction in
1st: $\theta_C, \theta_R, \theta_S$
2nd: $\theta_{TS}, \theta_{CS}$
3rd: $CL, CH$
sort classes $CL$ including characteristics $CH$
distinction in
1st: $\theta_{TS} \rightarrow CL_{TS} \rightarrow CH_{TS}$
2nd: $\theta_{TS} \rightarrow CL_{TS} \rightarrow CH_{CFG}^{ST}$
3rd: $\theta_{CS} \rightarrow CL_{CS} \rightarrow CH_{CFG}^{ST}$
4th: $\theta_{CS} \rightarrow CL_{CS} \rightarrow CH_{CFG}^{DY}$
rest: $\theta \rightarrow CL_{TS} \rightarrow CH_{TS}$
annotation weight coefficients
distinction in
1st: characteristics $WF_{A_{jC_i}}^{CH}$
2nd: classes $WF_{A_{jC_i}}^{CL}$
set of $CH_{TS}$ assorted descending by $WF_{A_{jC_i}}^{CH_{TS}}$
set of $CH_{CFG}^{DY}$ assorted descending by $WF_{A_{jC_i}}^{CH_{CFG}^{DY}}$
set of $CH_{CFG}^{ST}$ assorted descending by $WF_{A_{jC_i}}^{CH_{CFG}^{ST}}$
distinction in EE
1st: $TH$
2nd: $PO$
3rd: $PE$

Figure 72: Pre-process – set of decision variables

In the second step of the pre-process, we initialize and execute the aspect-based calculation methods in $MAS_C$. The utilization levels build the base of the power, temperature, and performance functions. We consider the utilization levels of the range between $[0,100]\%$ and provide the base values of each aspect concerning the initial configuration $\theta_C$. The aspect-based ranges have to be defined before the simulation loop starts to speed up the calculation of the energy efficiency ratio in relation to the optimization strategy.

We define the power range of the processor, as described in Equation (5.79), and store the results in lookup tables (LUT). We assume a linear relation between the utilization levels and the range of power consumption values. We consider the frequency range, see Equation (5.73), which we appropriately associate to the power range with a constant voltage. In addition, we specify the particular component states, which rely upon the available p-states $k$. The p-state is minimal when the utilization level is maximal, see Equation (5.168), which provides the extreme values, as shown in Equations (5.166) and (5.167). The voltage-frequency pair is specific to each p-state and defined by the configuration $\theta_C$.

$$u_{proc}(100\%) \rightarrow p - state(100\%) = 1 \tag{5.166}$$

$$u_{proc}(0\%) \rightarrow p - state(0\%) = k \tag{5.167}$$

$$p - state(u_{proc}) = \left\lceil -\frac{k-1}{100} * u_{proc} + k \right\rceil \tag{5.168}$$

We define the power range of the memory module, as described in Equation (5.68), considering the concept of [Mic 2007], see Equation (5.62). We group the major $IDD$ states to be considered into the idle state and the active state, which we further distinguish in the refresh mode or read-to-write mode. We define the boundary between the idle and active state by a utilization level at 20 percent[264] because the memory refreshes the data within the active state. We take the read-to-write ratio after reaching the 50 percent utilization level into account.

### 5.4.2 Simulation Loop

The aspect-based ranges of the initial configuration $\theta_C$ are the basis data of our energy efficiency calculation. We specify at each discrete time $t_k$ the energy efficiency ratio[265] $EE(\theta_C^l, \vec{U}_{t_k})$, whereby we assume that $t_k$ refers to a period $[t_k, t_{k+1}]$. We synchronize the simulation loop by a discrete time $t_k$, which the customer specifies in the workload scenario. We assume that the utilization levels are constant during our optimization process and calculate the energy efficiency ratio $EE(\theta_C^l, \vec{U}_{t_k})$ at each $t_k$. A special simulation case is the first optimization loop, wherein we calculate the energy efficiency ratio $EE_{BASE}$ considering the initial configuration $\theta_C$ and the decision variables. We set the local optimum to $EE_{BASE}$,

[264] Boundary at 20 percent: vendor-specific rule of thumb
[265] Ratio at $t_k$: a performance-to-power ratio, but if $\Delta t \rightarrow 0$ we assume the energy efficiency ratio

skip the step-based analysis, and start the alternation process. In the first loop at each $t_k$, we restrict the design space of the decision variables in dependence on the workload. We guarantee that we do not have a negative impact on the system performance so that the allocated time is always long enough to finish the job, as shown in Equation (5.109), when we adjust the characteristics. We remove the characteristics of the tree[266] that do not fulfill the requirements. We adjust the server configurations or characteristics by the discrete values specified as alternative configuration $\theta_C^l(x)$, whereby we analyze whether we have improved the energy efficiency. We update the configuration of the previous optimization loop and start the energy efficiency calculation. In the second optimization loop, we compare the actual energy efficiency ratio with the $EE_{BASE}$ and define the local optimum. We define the alternation of the decision variables, our optimization strategy, and the step-based analysis in Section 5.4.2.1. In principle, we iterate all possible decision variables of the design space, which is our final abort criterion. In consequence, we consider all subtypes of $\theta_{CS}$ and $\theta_{TS}$, which is a time-consuming process. We define a set of rules in our optimization strategy – when to increment $t_k$ after finding at least one local optimum[267]. We repeat the procedure of the remaining time and stop the simulation when we reach the end of the workload $t_k = T_u$. We provide a list of the best admissible solutions at each time $t_k$. We assume that a long-term efficient server configuration of the entire workload will save more energy than a locally optimized system. In the post-process of our simulation, we analyze the total workload scenario and sum up the time intervals of each sectional solution. At the end of our simulation, we define the global optimal solution by the configuration $\theta_C^l$, which dominates by the largest time. We do not consider the financial aspects, which the customer influences by the ordering process. The fastest return on investment (ROI) will be an additional criterion.

#### 5.4.2.1 Optimization Strategy

In the following section, we describe our *optimization strategy*, which specifies the procedure to alter the decision variables. In the interests of clarity, we aggregate the software-based as well as hardware-based configurations and characterizations in a *configuration* block. We abstract the internal communications and present them as a *process* block. Our simulation reacts upon the customer-specific workload scenario and considers the ambient temperature, which together with the *configuration* block builds the main input parameters. The *controller* observes the energy efficiency ratio, especially the power, temperature, and performance of each component $C_i$ of the time-based utilization levels. Our optimization strategy specifies the general procedure, which the controller takes into account. Figure 73 shows the simplified schematic of the *process* and *controller* within the simulation model.

[266] Remove characteristics: or generate a taboo list
[267] Local optimum: best admissible $\theta_C^l$ considering the adjusted decision variables

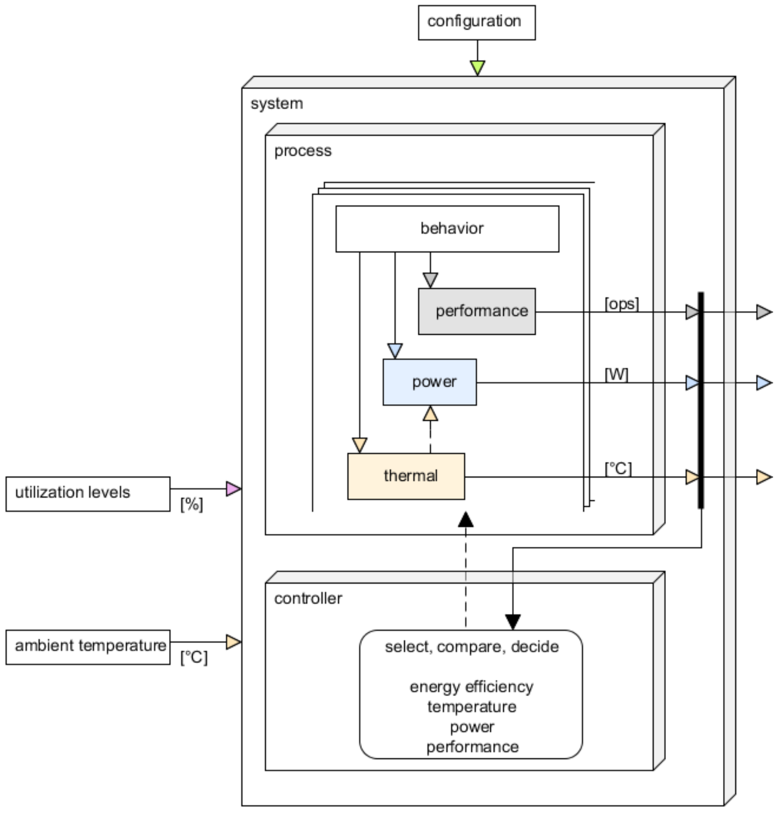


Figure 73: Simplified schematic of the system

Our aim is to derive the optimal component or system configuration during the design phase or the order process. We manipulate the server system $\theta_C$ within the system boundaries[268] on the basis of the results of the *step-based energy efficiency analysis*, which evaluates the energy efficiency ratio. We choose the decision variables in the *alternation strategy* and compare the optimization-based results. We define the abort criterion, which decides about additional alternations. The *controller* selects the adequate management technique with respect to the optimization strategy and the operational modus, which we further describe in the following section.

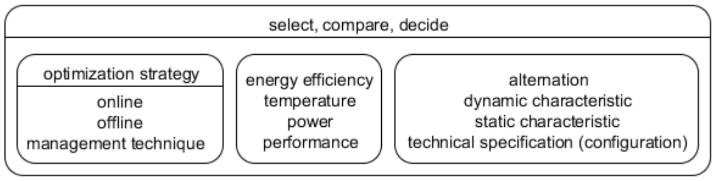


Figure 74: Controller – select, compare, and decide

[268] System boundaries: thermal limits $\eta$ or hardware constraints

In principle, our optimization strategy is based upon two cascading phases, which we differentiate into the short-term and long-term strategies. We guarantee in the short-term strategy, called the primary phase $(I)$, that our changes do not have any negative impact on the system or component's performance: i.e., we follow a purely greedy approach in the interests of simplicity. We may escape from local minima of the greedy approach when we use a metaheuristic algorithm. Kernighan-Lin, Simulated Annealing (SA), Evolutionary Algorithms (EA), or Genetic Algorithms (GA) are iterative approaches that seek a global optimum; either use an acceptable probability, or adjust the population[269]. In our simulation framework to reduce the risk of a local minimum, we do not specify an explicit algorithm.

We specify the primary phase as *online*[270] because we consider the short-term modifications when the server system is working. Herein, we prepare the most probable values of the decision variables to represent the reality, which form the input parameters of our optimization under most realistic conditions. We adjust the dynamic characteristics $CH_{CFG}^{DY}$ considering the thermal-based and power-based management techniques. We do not integrate our primary phase concept into a real server system, because we have to disable a couple of features and change the firmware that may result in an unstable server system. The internal system sensors limit the execution of our algorithm because of their latency and bus bandwidth. The embedded controller does not provide sufficient performance and storage capacity to execute our algorithm. Nevertheless, we assume an authentic and dynamic behavior of the server system in the primary phase as the basis of our optimization.

When the server system executes a workload for hours or days, the short-term management techniques are insignificant. We assume that an optimized static characteristic $CH_{CFG}^{ST}$ or an updated technical specification $CH_{TS}$ will save more energy in comparison to the short-term optimization. As a contrast to the *online* phase, we specify the secondary phase $(II)$ that is an *offline*[271] optimization, whereby the changes have indirect influences on the primary phase. We adjust the static characteristics or technical specifications, which the customer can manipulate only after shutdown or reboot the server system. Figure 75 shows the primary and secondary phases of our optimization strategy, which we outline in the following sections.

[269] Adjusting the population: selection, exchange, mutation, or recombination (crossover)
[270] Online: characteristic changes are possible during the runtime of the server system
[271] Offline: server system is shut down

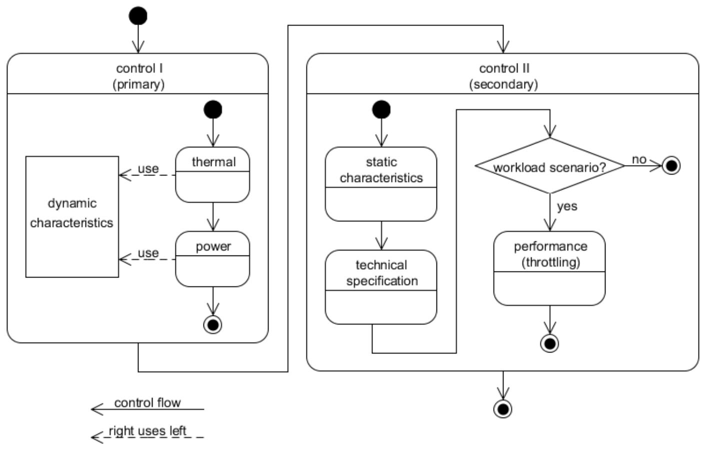


Figure 75: Optimization strategy phases

The authors in [FWB 2007] state that the proportional power consumption at all utilization levels, beginning at the low-intensive up to the high-intensive phase, is the major indicator to optimize the energy efficiency. Therefore, we concentrate upon the power optimization considering uncommon-case working conditions, such as low-intensive utilization levels, which the academic approaches neglect. Our aim is to decrease the power, meanwhile keeping the performance constant at the same time. Alternatively, we decrease the performance proportionally less than the power.

We specify the primary phase under the condition that the server system is active and executes a workload. In the first step $(I\text{-}I)$, we reduce the system temperature by enabling the dynamic thermal management (DTM) techniques, which does not affect any system performance up to when we exceed the damage temperature. The thermal control mechanism is a core feature of every server system that the system designer specifies in the design phase. The globally active mechanism monitors and controls the temperatures of the entire server system independently from the OS or utilization levels. The thermal control autonomously reacts upon an increasing temperature, which exceeds a predefined threshold and raises the cooling airflow according to the fully functional level of the server system. In industrial practice, the customer does not change the internal settings of the thermal control. A lower system temperature will increase the fan power, which is negligible in comparison to the necessary power of the HVAC system in the data center. We control the system and component temperatures within the reliability and functional level, which should not exceed the damage level. We react immediately when a component temperature reaches the critical threshold but considers a time delay when the warning threshold is exceeded. We are aware of the thermal limits to avoid the performance loss caused by the temperature. In both cases, we enable the thermal control, as shown in Figure 76.

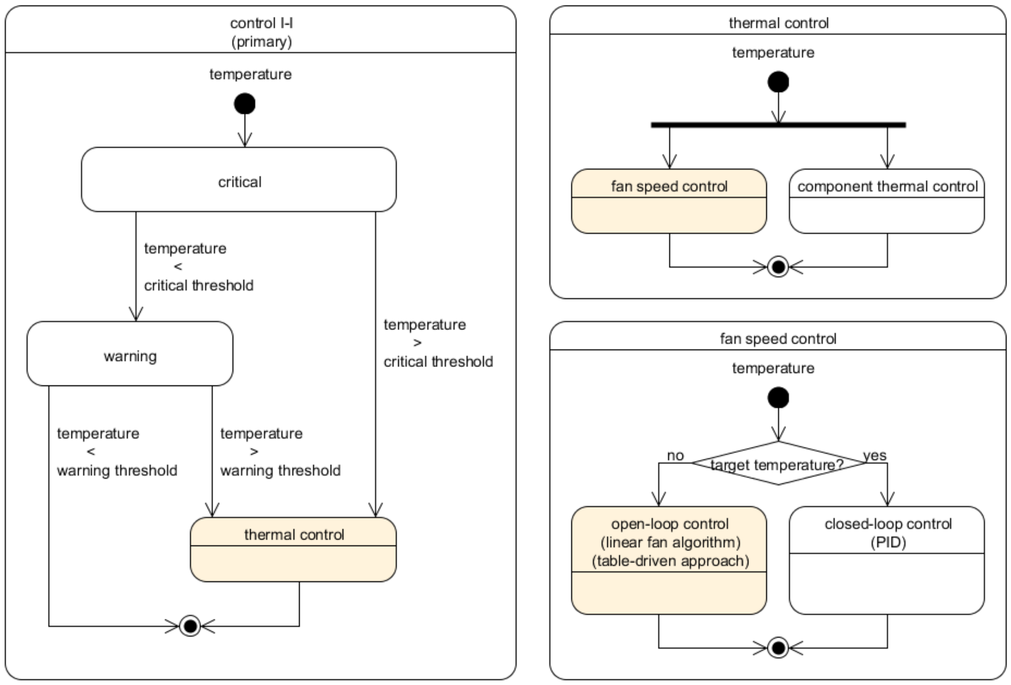

control I-I
(primary)
temperature
critical
temperature < critical threshold
warning
temperature > critical threshold
temperature < warning threshold
temperature > warning threshold
thermal control
thermal control
temperature
fan speed control
component thermal control
fan speed control
temperature
no
target temperature?
yes
open-loop control (linear fan algorithm) (table-driven approach)
closed-loop control (PID)

Figure 76: Thermal control in the primary phase

In the thermal control, we decide whether the fan speed control (FSC) or the component thermal control is an adequate strategy. In our primary phase, we neglect the component thermal control, which influences the performance because of the external limitations. Inside the fan speed control, we distinguish into the open-loop (linear fan algorithm, table-driven approach) and the closed-loop (proportional integral derivative – PID) control. In our prototype implementation, we consider the linear fan algorithm that does not require a continuous control process to a target temperature, see Figure 60.

In the second step $(I\text{-}II)$ of the primary phase, we reduce the power under the condition that the performance is not affected, such as a longer execution time. We decrease the fan speed to save power as long as the component temperatures are within the thermal limits to ensure the functionality, as done in commercial systems. We optimize the power after the temperature because of the management techniques that rely upon the OS. We reduce the average and peak power of the components by their dynamic characteristics, such as the voltage and frequency (DVFS), as autonomously done by every ACPI-based OS, either Windows or Linux, for example. We can reduce the frequency without decreasing the performance when the system is idle, which is independent of the specific OS. Consequently, we abstract the OS in our model and include the corresponding strategies that present the realistic behavior. Beneficially, the power consumption is less in comparison to the higher-frequency states, which results in a better energy efficiency ratio. We consider the efficient component states across the entire server system when we disable, power down, or set the sleep state of the unused components, as done in [RL2007]. We assume that the server system follows the DVFS and DTM strategies, which is a standard practice of the ACPI-based OS control. We summarize the time-sensitive DTM and DVFS strategies in $\gamma$, which optimize the worst-case power consumption in a time horizon of seconds or minutes. We consider the global system thermal management and the local power management technique at the same time, whereby the fraction of the fan power is less in comparison to the component power. On the basis of our analysis results, we change the management strategy to optimize the behavior within the primary phase. The authors of [FWB 2007] state that a holistic approach is more efficient than a successful local optimization of one component. We separately optimize each component but are aware of negative impacts on the global system.

We assume that the thermal as well as power management techniques work as described and consider them as the standard procedures in our simulation. If the approaches act differently in certain circumstances, we may adjust our primary phase to represent the common usage and behavior in a more realistic manner. We control the internal system temperature $(A)$ considering the ambient temperature in the dynamic thermal management and change the dynamic component characteristics $(B)$ considering the short-term strategies in our primary phase, see Figure 77.

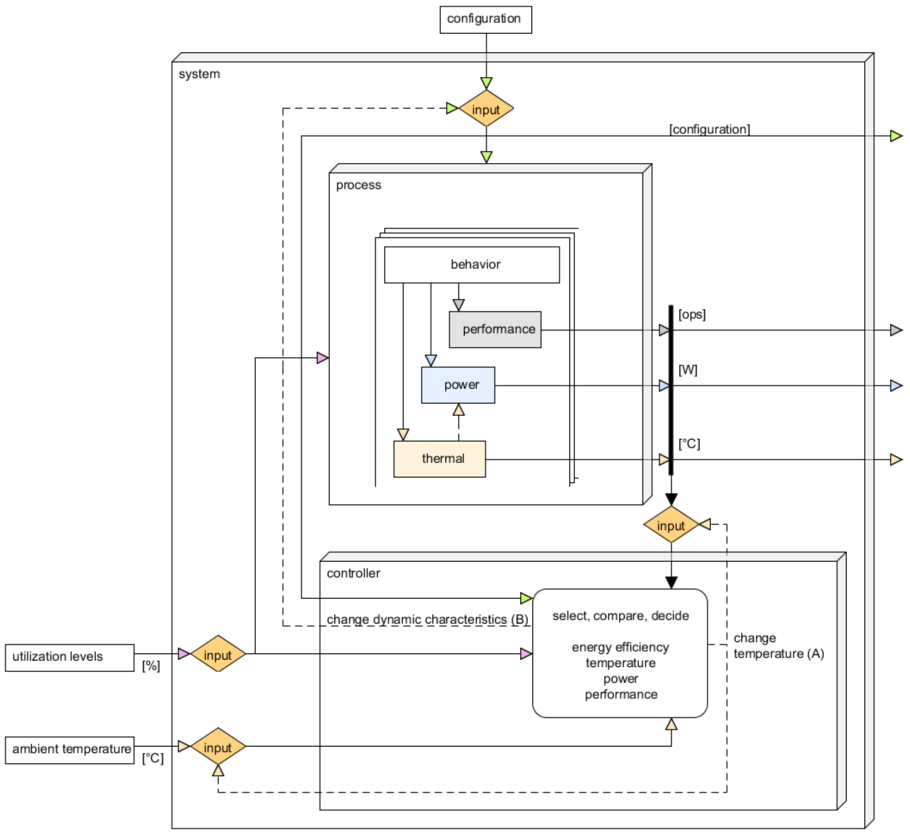


Figure 77: Primary optimization strategy

We move on to the secondary phase after finishing the adjustments of the primary phase. The following steps of the secondary phase concentrate upon the long-term strategies, which optimizes the server system in the early design phase. The most adjustments in the secondary phase are applicable when the system is off, especially a couple of the software-based settings $\xi$. In the first step $(II\text{-}I)$, we disable the features[272] before the system starts that do not result in performance loss. The customer usually does not disable the hardware-based features, such as the processor virtualization[273] support of the component, for instance. If the system is not in a virtualized environment, the processor virtualization support is negligible and we disable this feature to save energy. We adjust other BIOS/UEFI characteristics that are significant to the server system energy efficiency ratio but constant over hours or days, particularly in a data center. An example is the memory channel configuration, which defines the *operating* mode[274] of the memory modules.

[272] Disable features: alternatively adjust power-relevant characteristics
[273] Processor virtualization support: *AMD-V* or *Intel VT-x*
[274] Operating mode: independent, sparing (a reserved spare rank avoids failures, such as correctable errors), or mirroring (copy all data in an opposite channel to create redundancy)

We support the flexible adjustment of the hardware configurations, especially their static characteristics $CH_{CFG}^{ST}$, which the work of [RHW et al. 1995, Her 1998, GSI et al. 2002, MCE et al. 2002, CDS 2003, HSW et al. 2004, RL 2007] does not cover.

In the second step $(II\text{-}II)$, we vary the technical specification $CH_{TS}$ of the components which become relevant in the order process of the server system. We can improve the energy efficiency when we optimize the components concerning the specific demand. Our aim is to avoid under-utilized components and improve the non-peak efficiency considering the low-intensive workloads. The modern components execute the same workload in a more energy-efficient manner, which consumes less energy by performing the identical workload or provides more computation power (processing speed) with constant energy. In general, we prefer the energy reduction when we vary the technical specification, because we concentrate upon the actual performance demands and do not consider future demands. As part of our heuristic, we replace the components themselves, such as substitute a fully buffered dual inline memory module (FB-DIMM) by a registered module (RDIMM) which provides the same throughput with less energy. Furthermore, we consider the vendor and hardware generations when we alter or adjust the components to find the ideal server configuration. In this step, we are aware of the present utilization levels to avoid the full utilization or under-utilized components. Fewer chips on the module consume less energy, especially when the memory module is under-utilized: e.g., two fully utilized memory modules may consume more energy than twice the numbers under half of the utilization[275]. Consequently, we restrict or vary[276] the memory capacity upwards or downwards in dependence on the workload scenario. We assume a linear relation between the utilization levels and the total memory capacity. Subsequently, we update the utilization levels $(C)$ to be comparable between the diverse hardware variations. Figure 78 shows the abstract configuration changes of the static characteristics $(D)$ and technical specification $(E)$ in our secondary optimization strategy.

[275] Memory energy: power consumption of the memory modules is non-linear
[276] Vary the memory capacity: change capacity of a single module or the quantity of the modules

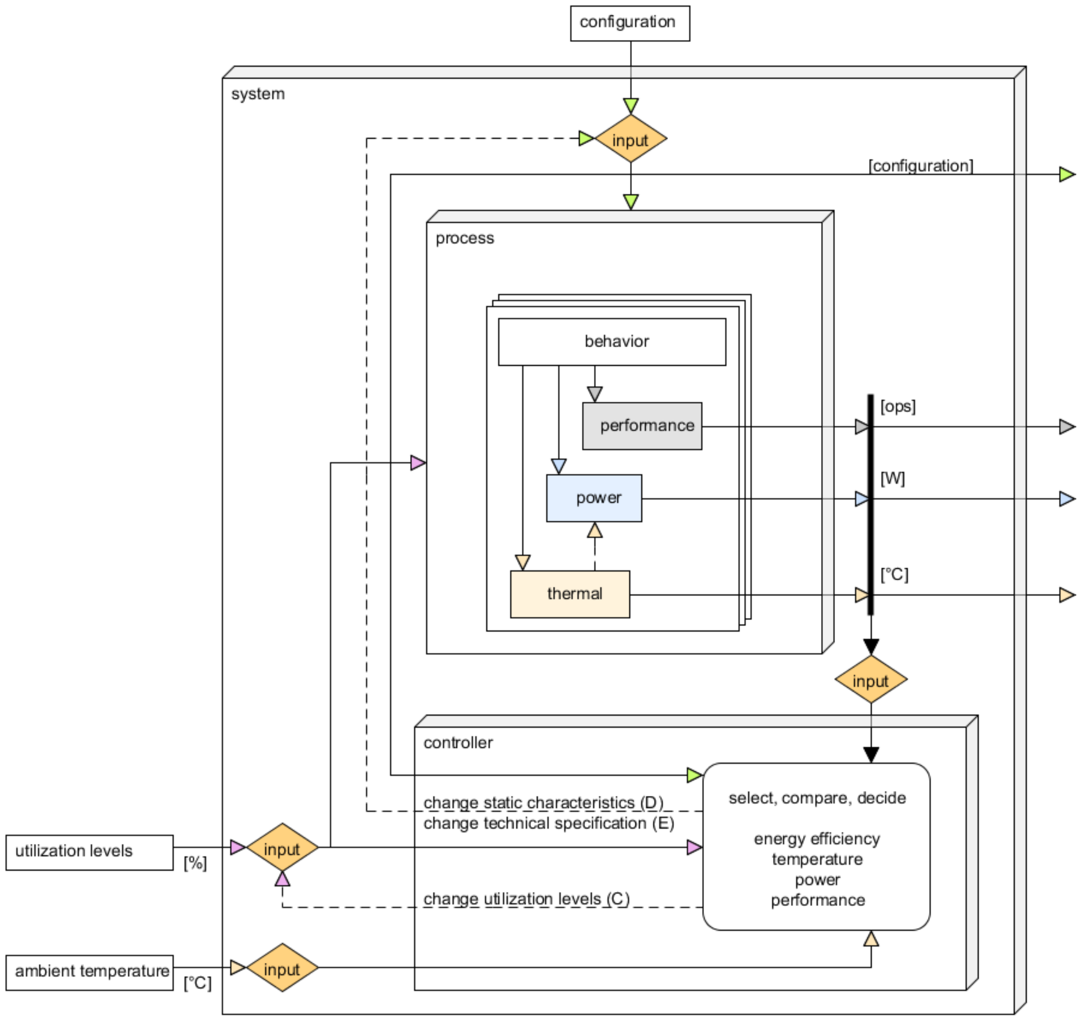


Figure 78: Secondary optimization strategy

Table 36 summarizes the adjustments in the primary and secondary phases of our optimization strategy.

**Table 36: Adjustments in the primary and secondary phases**

| | **Primary short-term, online strategy** | | **Secondary long-term, offline strategy** |
|---|---|---|---|
| $(\boldsymbol{I})$ | Dynamic characteristics $CH_{CFG}^{DY}$ $(B)$ | $(\boldsymbol{II})$ | Configuration |
| $(\boldsymbol{I}-\boldsymbol{I})$ | DTM $(A)$, component thermal management, fan speed control | $(\boldsymbol{II}-\boldsymbol{I})$ | Static characteristics $(D)$ $CH_{CFG}^{ST}$ |
| $(\boldsymbol{I}-\boldsymbol{II})$ | DVFS $(B)$ | $(\boldsymbol{II}-\boldsymbol{II})$ | Technical specification $(E)$ $CH_{TS}$ |
| | | $(\boldsymbol{II}-\boldsymbol{III})$ | Performance (throttling) on known workload scenario (post-process) |

In the previous section, we specify the cascading phases of our *optimization strategy*, which we realize in the alternation of the decision variables, as shown in Figure 68. Before we start the procedure, we iterative determine the energy efficiency ratio of the present server system and start the *step-based energy efficiency analysis*. We compare the energy efficiency ratios $EE$ and assume that the usefulness of our adjustments are represented by a scalar value of each aspect and finally in the energy efficiency ratio. We analyze the energy efficiency ratios of an iteration $(l)$ and the previous one $(l-1)$ to specify the impact on the basis of our adjustments between ${}_k\theta_C^l$ and ${}_k\theta_C^{l-1}$, as shown in the upper half of Figure 79. We sort the ratios $\{{}_k\theta_C^1, {}_k\theta_C^2, \dots, {}_k\theta_C^l\}$ at each $t_k$ in a list in descending order considering the improvement on $EE_{BASE}$. In the first optimization loop, the list covers only the $EE_{BASE}$ of $\theta_C$ as a local optimum. A larger value of $EE({}_k\theta_C^1)$ shifts the position of $EE_{BASE}$ in the list of the second-best solution[277]. The first entry into the list is more energy-efficient than the remaining energy efficiency ratios having a greater index, respectively the smaller indexes than $EE_{BASE}$ building the set of improved solutions at $t_k$.

[277] Position in the list: a larger index refers to the less optimal solution

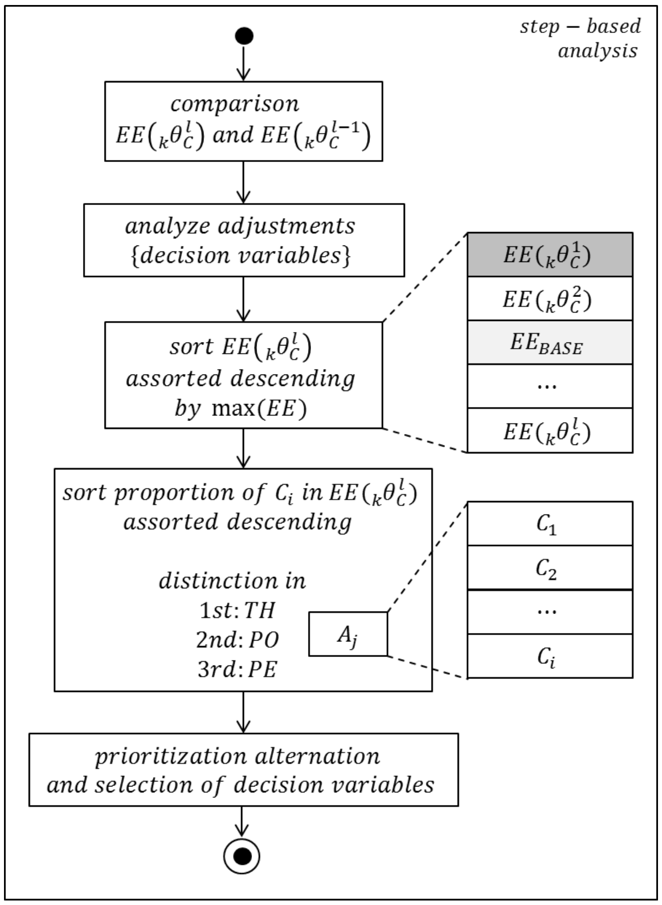

step − based analysis
comparison $EE({}_k\theta_C^l)$ and $EE({}_k\theta_C^{l-1})$
analyze adjustments {decision variables}
sort $EE({}_k\theta_C^l)$ assorted descending by max(EE)
$EE({}_k\theta_C^1)$
$EE({}_k\theta_C^2)$
$EE_{BASE}$
…
$EE({}_k\theta_C^l)$
sort proportion of $C_i$ in $EE({}_k\theta_C^l)$ assorted descending
distinction in
1st: TH
2nd: PO
3rd: PE
$A_j$
$C_1$
$C_2$
…
$C_i$
prioritization alternation and selection of decision variables

Figure 79: Step-based energy efficiency analysis

The alternation of all characteristics and configurations is too expensive in terms of simulation time and performance requirements. We require an adequate heuristic in the alternation strategy to specify the priority and selection of the decision variables. In the first instance, we specify the priority of the category in dependence on the workload scenario, which is processor-bounded, memory-bounded, or I/O-bounded. If we cannot determine the related category, we consider a default category that we assume as the highest-impact factor. If we know the specific benchmark represented by the workload, we predefine the significant category. We optimize the components that contribute to a lower amount by a lower priority only if needed or if an extremely high level of optimization is intended. Afterwards, we analyze the energy efficiency ratios related to the aspect-based sequence gained from the cascading phase of the optimization strategy. We start with the prioritized components to find the largest impact up to the lowest impact on each aspect $A_j$, which specifies the sequence of alternation, as abstractly shown in Figure 79.

As an example, we assume a processor-intensive workload that has the highest priority in our alternation because it has the most power-based impact upon the energy efficiency. We analyze the power consumptions of all categories and sort them by their values in descending order. The workload may change during the time, which may lead to another sequence of categories and components. We dynamically consider the categories by their flexible priority in our optimization strategy. We specify the alternation of the decision variables in the next section.

### *5.4.2.2 Alternation Strategy of the Primary and Secondary Phase*

We alter the decision variables on the basis of the cascading phases of the optimization strategy to provide the most probable presentation of the running server system, while adjusting all dynamic characteristics. We optimize the server system in the secondary phase on the basis of the changes in the primary phase, wherein we modify the static characteristics which require a repetition of the adjustments concerning the dynamic characteristics. If we adjust the static characteristic, such as the memory capacity, we respectively update the utilization level, as described in the second step of the secondary optimization phase. We modify the dynamic characteristic because of the changed conditions. In the next phase, we optimize the technical specification, which requires the alternation of the dynamic and static characteristics. In principle, we consider the hierarchical order of our configuration tree $\theta$ and use a bottom-up strategy to alter the decision variables beginning with the dynamic characteristics up to the static configuration, as shown in Figure 80.

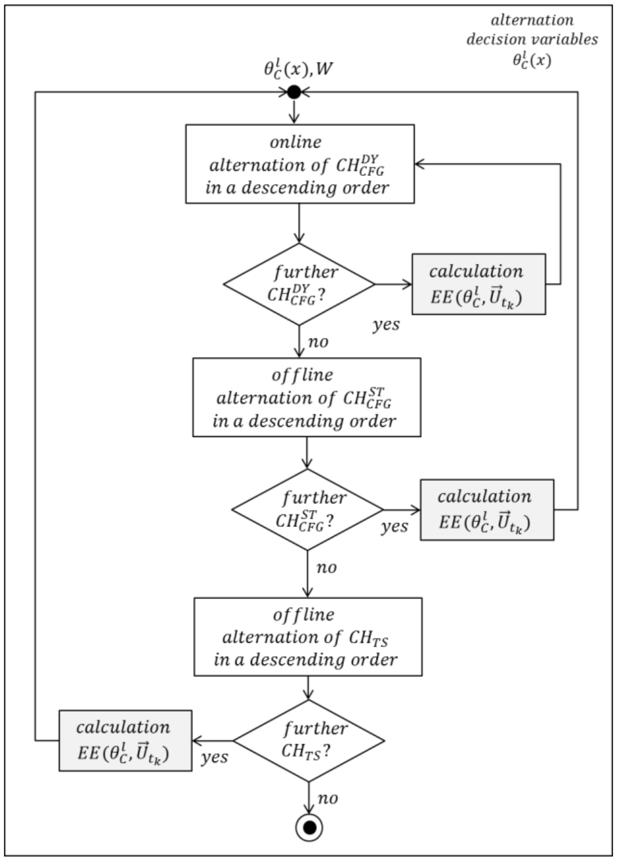

alternation decision variables $\theta_C^l(x)$
$\theta_C^l(x), W$
online alternation of $CH_{CFG}^{DY}$ in a descending order
further $CH_{CFG}^{DY}$?
yes
no
calculation $EE(\theta_C^l, \vec{U}_{t_k})$
offline alternation of $CH_{CFG}^{ST}$ in a descending order
further $CH_{CFG}^{ST}$?
yes
no
calculation $EE(\theta_C^l, \vec{U}_{t_k})$
offline alternation of $CH_{TS}$ in a descending order
further $CH_{TS}$?
yes
no
calculation $EE(\theta_C^l, \vec{U}_{t_k})$

Figure 80: Alternation of decision variables considering the cascading phases

As an aspect of the technical adjustments, we differentiate between high-sophisticated (enthusiast), midrange (standard), power-reduced, or performance-reduced components. A high-end component, such as a processor, wastes more energy while being idle in comparison to a midrange one, because of the larger base power. As one part of our strategy, we adjust the component by selecting a more efficient version. In addition, we consider the diverse vendors that support the same performance but consume less power because of their internal specification. After defining the category-specific sequence, we successively analyze the relevance of every characteristic ordered by their impacts, which is a dynamic process.

We have to consider any characteristics and configurations in the entire design space, whereby we explore single decision variables or multiple combinations. We can guarantee to find the best solution after completely traversing the configuration tree considering all possible characteristics and configurations. We define a heuristic to avoid the disproportionate increase in the alternation complexity and the corresponding simulation time, which reduces the design space and provides flexible abort criteria. We specify the knowledge-based and vendor-specific alternation rules that exclude irrelevant adjustments, such as insignificant modifications, and define a preference for the decision variables to reduce the optimization effort as well as speed up the procedure. Our aim is to prioritize the characteristics and configurations of each component concerning their impact upon the energy efficiency ratio.

In our alternation strategy, we consider the separation of the aspects as well as components in relation to the workload gained from the *step-based analysis* and priority sequence of the pre-process. We apply the annotation of the weight coefficients $WF_{A_{j_{C_i}}}^{CH}$ of each characteristic[278] in each function $F_{A_{j_{C_i}}}$ at $MAS_C$ and use the descending list of the weight coefficients to determine their impact on the total energy consumption, for instance. The highest weight coefficient has the most impact on the calculation function, whereby we determine the priority to the related characteristic concerning the list index. We restrict the optimization by setting the abort criterion when the remaining characteristics have less influence than the present ones. We assume a set of characteristics $CH = \{CH_1, CH_2, \dots, CH_N\}$ and their corresponding weight coefficients $WF_{A_{j_{C_i}}}^{CH}$ of any aspect $A_j$. Table 37 exemplarily presents possible priorities to the characteristics and relative impacts on the range between zero and one of the unspecific aspect.

[278] Specification of the weight coefficients: see Figure 53 and Figure 54

Table 37: Prioritization characteristics by their weight coefficients

| Priority | Weight coefficient $WF_{A_{j_{C_i}}}^{CH_l}$ | Relative impact on $A_j$ of component $C_i$ | Characteristic $CH_l$ |
|---|---|---|---|
| ***Critical*** | $WF_{A_{j_{C_i}}}^{CH_3}$ | 0.40 | $CH_3$ |
| ***High*** | $WF_{A_{j_{C_i}}}^{CH_1}$ | 0.35 | $CH_1$ |
| ***Medium*** | $WF_{A_{j_{C_i}}}^{CH_4}$ | 0.15 | $CH_4$ |
| ***Low*** | $WF_{A_{j_{C_i}}}^{CH_2}$ | 0.10 | $CH_2$ |

Weight coefficients: annotation, see Figure 53 and Figure 54

From this table, we would conclude to optimize the *critical* characteristic $CH_3$ that has the most single impact by approximately $0.40$ on the aspect $A_j$. The sum of the remaining weight coefficients $\sum WF_{A_{j_{C_i}}}^{CH_l}, l = 1,4,2$ is larger than the weight coefficient of the *critical* characteristic $CH_3$. Therefore, we consider the second largest (*high*) priority $CH_1$ in our optimization strategy. We abort the alternation when the sum of the remaining weight coefficients (relative impact) is smaller in comparison to the actual weight coefficient determined by the priority list. In our example, the sum of the weight coefficients $\sum WF_{A_{j_{C_i}}}^{CH_l}, l = 4,2$ is smaller than the weight coefficient of the characteristic $CH_1$. We restrict the design space of each category on the basis of the impacts on the dynamic characteristics, static characteristics, or technical specification. Accordingly, we dynamically annotate the classes and characteristics of the tree in relation to the actual management technique on the basis of the aspect. The abort criteria restrict the design space and in consequence we do not consider all characteristics in the configuration tree. If the simulation time is uncritical, or we require a higher accuracy, we will neglect the abort criteria and consider the remaining characteristics.

We avoid superposition during our iterative process and select the decision variables in dependence on their weight coefficients in the ranked list sorted by their impacts. When we change one decision variable[279] in a system-compatible solution ${}_k\theta_C^l$, we mean an adjustment of the value of the predefined range considering the thermal constraints, for instance. We extend our approach with regard to the coupled characteristics, which influence each other. We can also reduce the complexity of the design space when we alter only the decision variables of ${}_k\theta_C^l$ that are better than $EE_{BASE}$, but this may lead to ignore alternative optimal solutions.

[279] Change one decision variable: try to keep the remaining decision variables unchanged

### 5.4.3 Post-Process

At the end of each optimization loop at $t_k$, our step-based analysis provides a ranked list in descending order that specifies the set of optimal solutions when the present energy efficiency is better than $EE_{BASE}$. In the global analysis, we intuitively select the optimal solution $\theta_C^l$, which dominates for the longest period of time. The optimal solution represents a rough approximation, which does not fit if the energy efficiency has an exponentially high value for a short time but an extremely low value for a long time. Consequently, the dominant energy efficiency ratio would be very inefficient in such a situation. To be more precise, we optimize the balance of the power and respective energy in the integrand of the time integral. We consider the energy efficiency ratios and their aspect-based values, which we weight by their corresponding time, and minimize the integral at $T_u$. Herein, we provide a purely technical optimal solution, but do not consider financial aspects like the fastest return on investment (ROI), which the customer influences during the ordering process.

We can handle an additional optimization when we consider the complete workload scenario. A high throughput of the component may proportionally increase the temperature and power consumption of the entire time. We analyze the various benchmarks, their parameters, and the global settings to optimize the components. Herein, we decide whether we can create bottlenecks, which will be compensated by fewer utilization levels in the following time step. As an additional step of the secondary optimization strategy ($II$-$III$), but established in our post-process, we restrict the performance, called throttling, and respectively the power consumption in dependence on the complete workload scenario. We limit the maximal utilization levels and avoid the full utilization level, which results in a reduced performance of the remaining components because of additional waiting cycles. We restrict the read-write bandwidth in the memory throttling technique[280] that specifies the highest utilization levels to tailor the traffic to the workload demand. Alternatively, we define the theoretical maximal transfer rate to save energy because the modules do not reach their largest throughput. The authors of [Bel 2000] investigate on the processor throttling strategies to keep the system in a predefined power range, which limits the frequency but may exceed the interval while executing the job. Their event-driven approach controls the component activities of each thread-specific event. We are not able to adjust the events, because of the missing information about the explicit hardware counters in the early design phase.

## 5.5 Summary

In our concept, we support the vendor and customer perspective to calculate the energy efficiency of a server system. We consider a steady workload, as done in the commercial tools, which we distinguish between the diverse component-bounded utilization levels. We specify the utilization levels as profiles that form in close conjunction our workload scenario. The

[280] Throttling restriction: limit the rate of accesses

scenarios create a realistic image of the application software in the *externals*. For the purposes of the multi-aspect server simulation, we transform the steady workload to continuous values to optimize the energy efficiency in the long-term (static configurations, technical specification). As part of our optimization strategy, we consider the common approaches, such as DVFS and DTM, to represent the short-term behavior (dynamic characteristics). We specify the cascading primary (*online*) and secondary (*offline*) phases to alter the relevant characteristics and configurations. In the *configuration and characterization* layer, we identify the system components, characterize the models, and define the aspect-based functions to calculate the energy efficiency. Herein, we consider the design implications, such as spreadsheets, analysis results, or vendor-based measurements. We assume the relations between the aspects of a single component and in the entire server system. We define the rules and equations of the controller in the *characterization* activity, while the controller is executed in the *simulation* activity. In our concept, we optimize a server system at each time step when the utilization levels change. We analyze the actual results and decide on the adequate management technique in the optimization strategy, according to the two optimization phases. We analyze the impact of our changes concerning the energy-to-performance ratio in comparison to the base energy efficiency of the initial server configuration. The adjustments of the server configuration and characterization require an additional calculation, which results in an iterative approach. We decide on our optimization strategy whether no further alternation is possible or required. Afterwards, we consider the subsequent part of the workload until the end of the simulated time. We select the globally optimized server configuration and characteristic that dominates for the largest time. Figure 81 shows a brief overview of the basic constituents of our five-step concept.

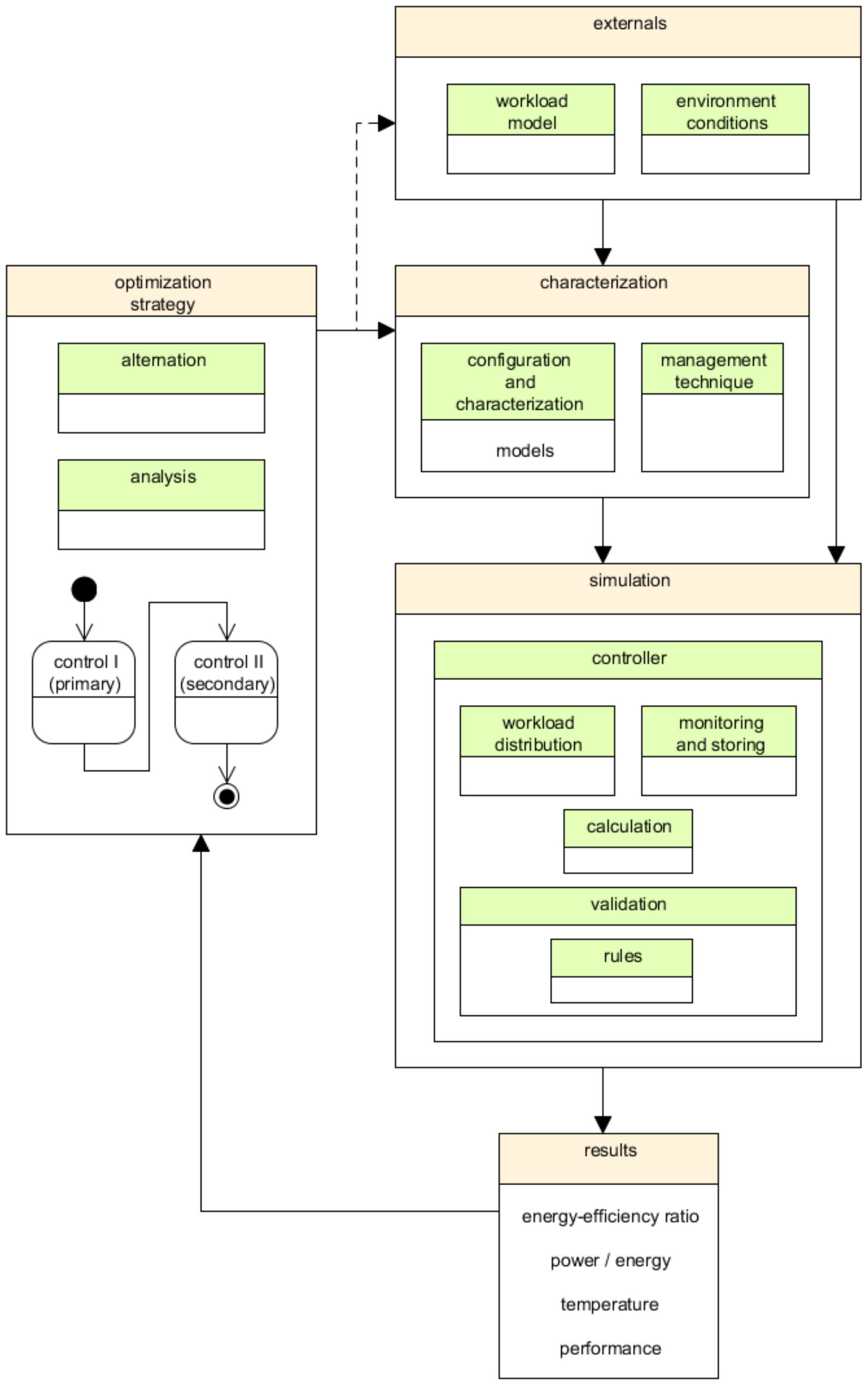


Figure 81: Five-step concept including basis constituents

# 6 Design and Implementation of the Architecture

We develop the simulation framework to calculate the energy efficiency of a server system with respect to a customer-specific workload on the basis of the concept described in the previous section. We abstract unnecessary details as used in a couple of academic models. In contrast to most of the commercial approaches, we extend the level of detail by certain characteristics and explore them to power, temperature, as well as performance limits. We consider fundamental restrictions and constraints on power, energy, performance, and temperature under realistic assumptions. The aim of our optimization system is to provide an optimized server system that operates in an energy-efficient corridor. For this purpose, such values of the decision variables $x = \{\theta_C, CH_{TS}, CH_{CFG}^{ST}, CH_{CFG}^{DY}\}$ are selected that maximize the performance and minimize the power (energy).

We specify the simulation model in MATLAB[281] and perform the continuous simulation using Simulink[282]. Our simulation framework includes the data processing, which provides the mathematical equations to calculate the multi-aspects of each component. We implement the non-linear behavior, which reacts to the time-dependent utilization levels by numerical integration of a system of ordinary differential equations (ODE). We apply numerical methods provided by MATLAB to solve the equations. When we have to predict the power consumption of a next-generation component, we use various interpolation methods. The customer specifies the simulation input in discrete[283] but not equidistant time steps, which results in changes of the simulation state at discrete instants of time. On the other hand, Simulink steps through each time interval $T_S$ using a fixed sample time on a 1-second basis[284]. We include the thermal development over time and thus we obtain a continuous-values simulation model, including feedback loops.

We realize our concept in MATLAB/Simulink using the common *Model-View-Controller* (MVC) approach, which is a three-layer design pattern to separate the major functionality. A change of any layer does not necessarily affect the remaining ones. This facilitates maintainability. We encapsulate the processing and storage activities to ensure integrity and consistency among the data. Our text-based and loosely coupled design provides a high level of flexibility and scalability for implementing further server generations considering dynamic structures and customizable weight coefficients. A designer or customer can easily extend or update the data of the simulation model.

[281] MATLAB: implementation in *.m scripts or functions
[282] Simulink: implementation in *.mdl file
[283] Discrete and continuous time or values: see Section 3.2
[284] Sample time: update the basis with regard to the time interval

We specify the components and its aspect-based calculation methods in MATLAB within the *model layer*, but represent the components in Simulink as abstract blocks of the *controller layer*. The *controller layer* assigns the input to the components, such as the workload, and calls the aspect-based functions of each time step. The Simulink environment monitors/controls the relations of each single component and the behavior between them. In the *controller layer*, we analyze the results and make logical decisions, such as optimization, whereby we follow our alteration strategy. In the *view layer,* we provide a graphical user interface (GUI) for the customization of the workload and the configuration of the server system. Using the GUI, we start the simulation and update the graphical elements to visualize the simulation results.

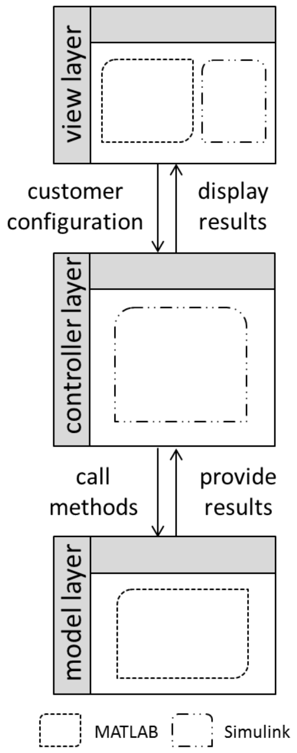


**Figure 82: Model-View-Controller (MVC) using MATLAB/Simulink**

The interaction between MATLAB and Simulink is done through callback functions, a listener, and workspace variables, but MATLAB and Simulink have their own internal representations of the variables. Consequently, we extend our MVC approach in order to enable cross-border exchanges of data[285]. We implement the *event_listener()* function to receive the GUI-based data and transfer it into the Simulink environment. When the simulation has finished, we update the GUI of MATLAB by using the *update_GUI()* method. The *set_param()* and *get_param()* functions are default communication methods, which we partly use in our initialization phase. Figure 83 briefly shows the communication between MATLAB and Simulink.

[285] Data exchange: possible alternative functions are *assignin*(), *set*(), or *setappdata*()

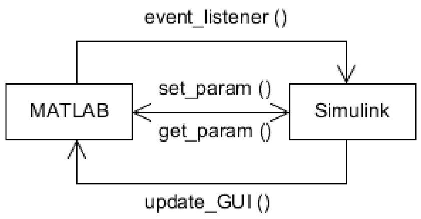


Figure 83: Communication between MATLAB and Simulink

## 6.1 View Layer

We implement the *view layer* for the representation of results and for supporting the customization of the input data, such as the external environment or the application software. We implement a dedicated MATLAB GUI that consists of the $(i)$ the workload configuration, $(ii)$ the server configuration, and $(iii)$ the ambient temperature settings[286], which build the primary input data of our simulation model. We assume that an efficient cooling outside the server system exists and no external influence may change the temperature: i.e., we assume the ambient temperature as a constant input. Furthermore, in our GUI we include $(iv)$ the start-stop functionality of the simulation and $(v)$ a graphical area for the representation of the simulation results[287]. Figure 84 shows the schematic view of the GUI, and Figure 85 illustrates the MATLAB implementation.

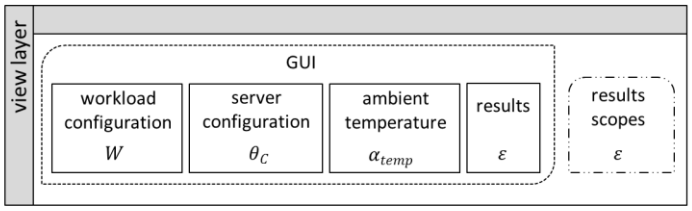
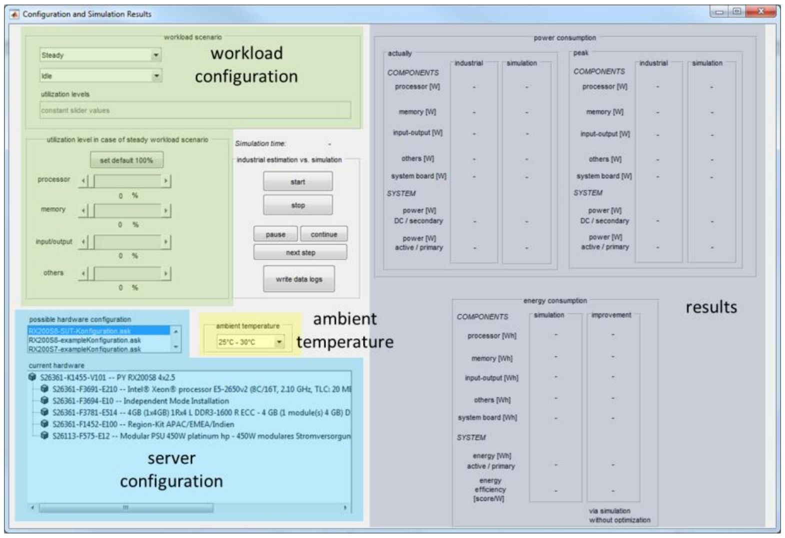

view layer
GUI
workload configuration $W$
server configuration $\theta_C$
ambient temperature $\alpha_{temp}$
results $\varepsilon$
results scopes $\varepsilon$

Figure 84: View layer – schematic GUI

[286] Ambient temperature $\alpha_{temp}$: we consider further environment settings at the *model layer*

[287] Simulation results: partly visualize in MATLAB GUI and Simulink scopes



RX200S8-SUT-Konfiguration.ask
RX200S8-exampleKonfiguration.ask
RX200S7-exampleKonfiguration.ask

S26361-K1455-V101 -- PY RX200S8 4x2.5
S26361-F3691-E210 -- Intel® Xeon® processor E5-2650v2 (8C/16T, 2.10 GHz, TLC: 20 MB
S26361-F3694-E10 -- Independent Mode Installation
S26361-F3781-E514 -- 4GB (1x4GB) 1Rx4 L DDR3-1600 R ECC - 4 GB (1 module(s) 4 GB) D
S26361-F1452-E100 -- Region-Kit APAC/EMEA/Indien
S26113-F575-E12 -- Modular PSU 450W platinum hp - 450W modulares Stromversorgun

**Figure 85: MATLAB GUI implementation**

The customer directly specifies the workload, designates the ambient temperature, and selects the concrete server system by using the GUI. We assign an explicit handler (callback function) to every MATLAB GUI event[288] to transfer the GUI-based data to our *model layer*. We visualize the initial customer-specific configuration and show further details, which we require in our optimization process. We graphically present the power and energy consumption at the GUI, which is comparable with the commercial tools in industrial practice. We initialize the output fields by a hyphen, update the GUI when the results are available, and visualize the related energy efficiency ratio to additional scopes[289] with respect to the continuous time domain. We provide the optimized server system configuration in a text-based format.

### Server System Configuration and Workload Configuration

We require an individual server system characterized by a configuration file[290]. Our simulation framework manages a server configuration, which we specify in a commercial tool called *PC- / System Architect*[291] developed within Fujitsu. This external tool has been designed as a common means to configure and order a server system manually. In principle, we can support diverse configuration tools, but we have to adjust our parsing algorithm to be compatible with such tools, e.g., the Dell *Energy Smart Solution Advisor*[292]. The Fujitsu tool stores one or more rack chassis in a single file, including multiple, heterogeneous, and rack-based server systems[293].

[288] Event: mouse clicks of a button, slider, or scroll bar
[289] Scope: graphical elements in Simulink (plots)
[290] Configuration file: proprietary format (*.ask), text-based
[291] *Fujitsu PC-/ System Architect*: http://configurator.ts.fujitsu.com/public/
[292] Dell Energy Smart Solution Advisor: http://essa.us.dell.com/DellStarOnline/DCCP.aspx
[293] Server systems: blade-based servers are also possible, which we neglect

The complexity of the architecture as well as of communication exponentially increases if a complete rack chassis is considered. In our prototype implementation, we concentrate on a certain rack-based server system architecture, which we exemplarily simulate and optimize.

In addition to the configurability, we support the intended flexibility of the workload to realistically reproduce and simulate the behavior of a server system. Our aim is to represent an hourly up to daily server usage and create an image of the customer-specific application software. We want to be independent of specific benchmarks, but support synthetic benchmarks in an abstract manner. In principle, our workload model is compatible with the most of commercial tools. In addition, we provide variations over an individual timespan. We abstract from specific instructions because the customer cannot predict them. We consider the influence of the OS type[294] simply by a factor. Our challenge is to overcome the workload limitations, as described in Section 4.1. In the next section, we describe the workflow of the customer-specific workload configuration, as shown in Figure 86. The customer selects either a steady or continuous workload scenario in the GUI, whereby the steady workload defines constant utilization levels over the complete simulated time.

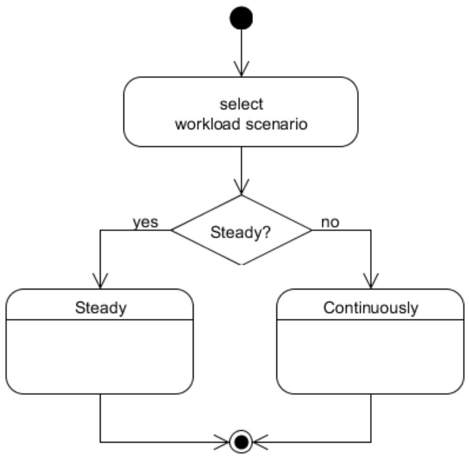


**Figure 86: Workload configuration – workflow**

In the steady case, the customer selects a single pre-defined workload profile as a steady workload, which is a common approach within commercial tools. In compliance with several industrial tools, we specify the following profiles:

- *Idle*
- *Transactional*
- *Computational*
- *Memory-intensive*
- *Worst-case*

[294] OS type: integrated in the optional settings of the application software

The *idle* workload considers the lowest $(0\%)$ utilization levels and the *worst-case* represents the highest $(100\%)$ ones of all components within the entire server system. The *transactional* profile is a synonym for an I/O-bounded workload and reflects applications such as VMWare[295], SAP[296], or Java-based applications. We subsume the high-performance computing and processor-bounded workloads in the *computational* profile. The *memory-intensive* profile refers to database applications (SQL server) or exchange servers. In our *model layer*, we distinguish between read-intensive and write-intensive applications. In accordance to most of the academic approaches, we express customization by a read-write-ratio. We specify the default utilization levels of every profile in an extra configuration file and initialize them, after a customer selects a steady workload. The customer can modify the utilization levels of each category by changing the defaults or adjusting the position of specific workload sliders. An extra button provides the opportunity to set all utilization levels to worst-case. We load the steady workload into the Simulink environment after generating a continuous and time-based stimulus.

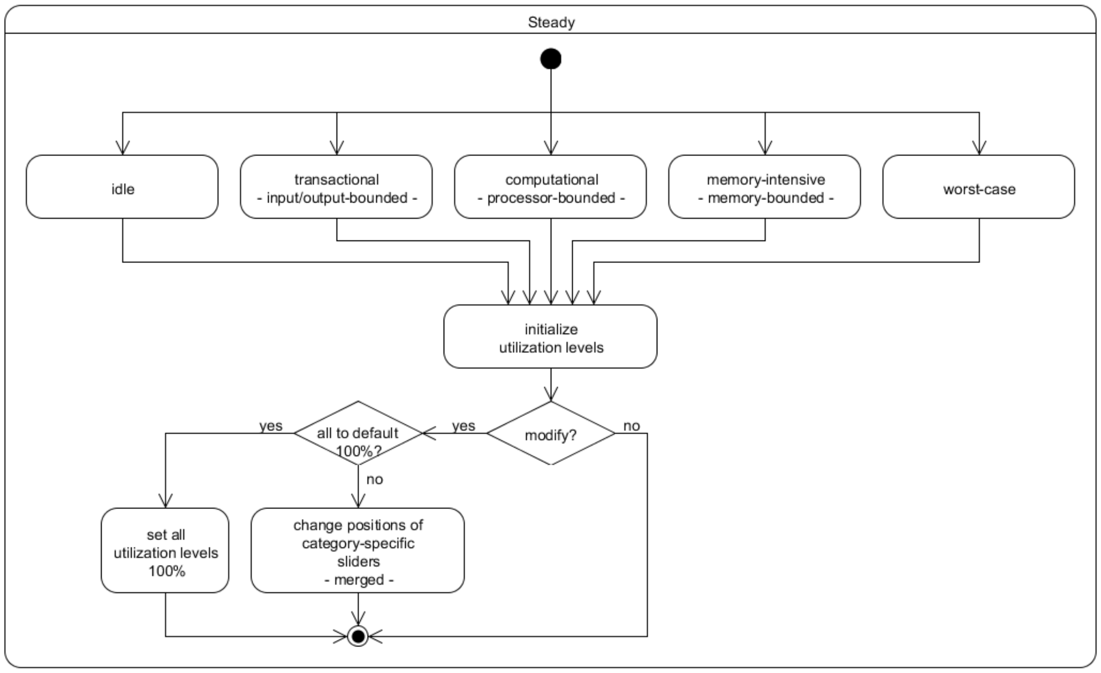
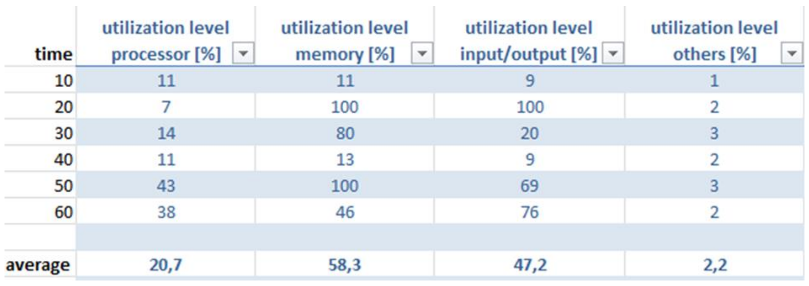

Steady
idle
transactional - input/output-bounded -
computational - processor-bounded -
memory-intensive - memory-bounded -
worst-case
initialize utilization levels
modify?
yes
no
all to default 100%?
yes
no
set all utilization levels 100%
change positions of category-specific sliders - merged -

Figure 87: Steady workload – workflow

As an alternative to the steady workload, we support a continuous workload that builds a more realistic use case. We define the continuous workload in a customizable file[297] that is composed of multiple utilization levels over a timespan. The columns specify the utilization levels of each category in the interval $[0,100]$ and the rows define discrete integer times. We

[295] VMWare: http://www.vmware.com/
[296] SAP: http://go.sap.com/index.html
[297] Customizable file: Excel file (*.xlsx), settings specify the used rows and columns

automatically parse the configuration file, convert the data into MATLAB-compatible format and check the input data. We expect that the customers manually specify the utilization levels in the configuration file, which is a common use case of industrial practice. Figure 88 exemplarily shows a workload scenario that has equidistant time steps; however, we also support non-equidistant time steps when a customer specifies a complete day or week. In the example, the server utilization is assumed to be extremely low at early-morning hours and would increase at office hours, as described for an SVN server in Section 3.3. We expect that the customer will provide the average utilization levels with respect to the opening and closing hours, which limits the access to the SVN server at a certain time. We consider constant utilization levels until another value is provided. In reality, the utilization levels fluctuate within milliseconds or microseconds, which we cannot consider. Therefore, we average the values over a ten-second base.



| time | utilization level processor [%] | utilization level memory [%] | utilization level input/output [%] | utilization level others [%] |
|---|---|---|---|---|
| 10 | 11 | 11 | 9 | 1 |
| 20 | 7 | 100 | 100 | 2 |
| 30 | 14 | 80 | 20 | 3 |
| 40 | 11 | 13 | 9 | 2 |
| 50 | 43 | 100 | 69 | 3 |
| 60 | 38 | 46 | 76 | 2 |
| | | | | |
| average | 20,7 | 58,3 | 47,2 | 2,2 |

Figure 88: Customer-specific workload scenario

In addition to the workload configuration, we check a plausibility of the utilization levels and automatically correct the values when we find an unrealistic assumption. For instance, a customer specifies more than one category by a utilization level of $100\%$, see Section 3.4. In our example, we fully utilize the memory and I/O at $t_k = 20s$ up to $t_{k+1} = 30s$. We calculate the average usage of both categories to define which category is more dominant in our timespan. We limit the I/O utilization level at a value of $80 - 90\%$ based on the rule of thumb, gained from the empirical measurements in a real system. We graphically show the current utilization level according to our correction and inform the customer about the correction with a warning message in the MATLAB environment.

## 6.2 Model Layer

We assemble the functional description of the components in a modular and isolated manner to ensure scalability. The model layer is the core concept of our simulation framework. It centrally provides the mathematical background and is used to calculate the aspect-based values in the *controller layer*. Our framework provides the availability to add next-generation components and server systems at various abstraction levels. Using a decoupled and hierarchical concept, we realize the abstraction between the aspect-based component description, the behavior, and the entire system. We separately define every component, which we may reuse in several server types, but influence the behavior using a higher-level specification in order to represent unique behavior in different compositions. We abstract from unimportant features and details of the server system such as to neglect operations that are executed once, as done in [BHS 1998].

We can define any kind of abstraction level of a component, but we exemplary concentrate upon utilization-based procedures. We define our category-specific methods, which we decoupled from the exact data to be universal and independent from the specific methods. We implement the logic that we require on the transition from the utilization-based calculation methods to a higher level of detail, such as power states or certain instructions. We consider external settings within our mathematical definitions, use MATLAB algorithms, and include a database. The database stores the possible values of the *server system characterization and configuration* containing the default utilization levels of our workload profiles, the weight coefficients, certain constraints, and the values of the configuration tree. We obtain high-quality data from the database on the basis of the server type and its compatible components, which we gain from real measurements and different data sources[298]. We require additional data, which we specify as metadata to realize our concept. Table 38 exemplarily[299] shows the relevant metadata of a memory module, which we consider in our aspect-based calculation methods.

**Table 38: Memory module $8GB\ (1x8GB)\ 2Rx4\ L\ DDR3-1333\ R\ ECC$ – additional metadata**

| Memory characteristics | Metadata |
|---|---|
| **Technical configuration** | |
| Vendor | *vendor="Samsung"* |
| Family | *fam="240 Pin DDR3 DIMM"* |
| Die (component revision) | *die="D"* |
| **Static characteristics** | |
| Capacitors / capacitance, capacity (size) [GB] | *cap="8GB"* |
| Density [GB] | *dens="8GB"* |
| Fabrication size [nm] | *nm_tech="44nm"* |
| Synchronization mode | *sync="registered"* |
| Module ranks, rank linking (data width) | *ranks="2R"* *ranks_x="x4"* |
| Timings | *cycle_time="1.5ns", cas="CL9"* |
| Resistance | *rth="56°C/W"* |
| **Dynamic characteristics** | |
| Frequency [MHz] | *MHz="667", frequency=MHz* |
| Voltage | *volt="LV", LV="low-voltage"* |
| Error correction | *corr="ECC"* |

[298] Data sources: diverse benchmarks, spreadsheets, wide ranges of sensor data, or vendor-specific data
[299] Example of a memory module: table contains metadata which we can consider in the order process

In addition, to customize our framework we define several default settings, which are fixed for the simulation runs. We specify the location of our database, the server configuration file, the energy units, or the supply voltage, for instance.

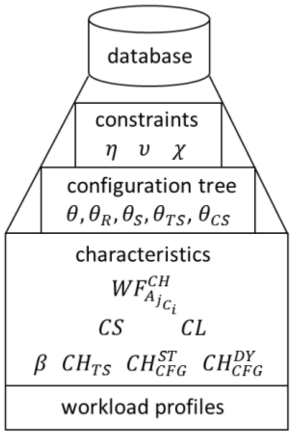


**Figure 89: Database**

The key constituents of the model layer are the mathematical descriptions of the components, as shown in Figure 90. We separately specify the components and their explicit calculation methods of each aspect, whereby we additionally specify the individual behavior. At the next abstraction level, we define the relations between the aspects and the components. We implement a function of each aspect in relation to the utilization level and specify a lookup table that represents the values of $10\%$ step-based utilization levels, as common is in industrial practice. We consider linear and non-linear interpolation methods provided by MATLAB to obtain the values with a smaller step size. The temperature itself directly depends upon the power consumption. Therefore, we do not need to define it on a stand-alone basis. To specify the energy efficiency ratio, we normalize the performance scores and power values being compatible to the simulation and having the same range of values. To improve the energy efficiency ratio, we consider the results and implement the *optimization strategy*[300], considering the two cascading phases, see Section 5. Herein, we specify the related methods and apply the existing MATLAB algorithms to optimize an entire server system.

[300] Optimization strategy: includes the management techniques

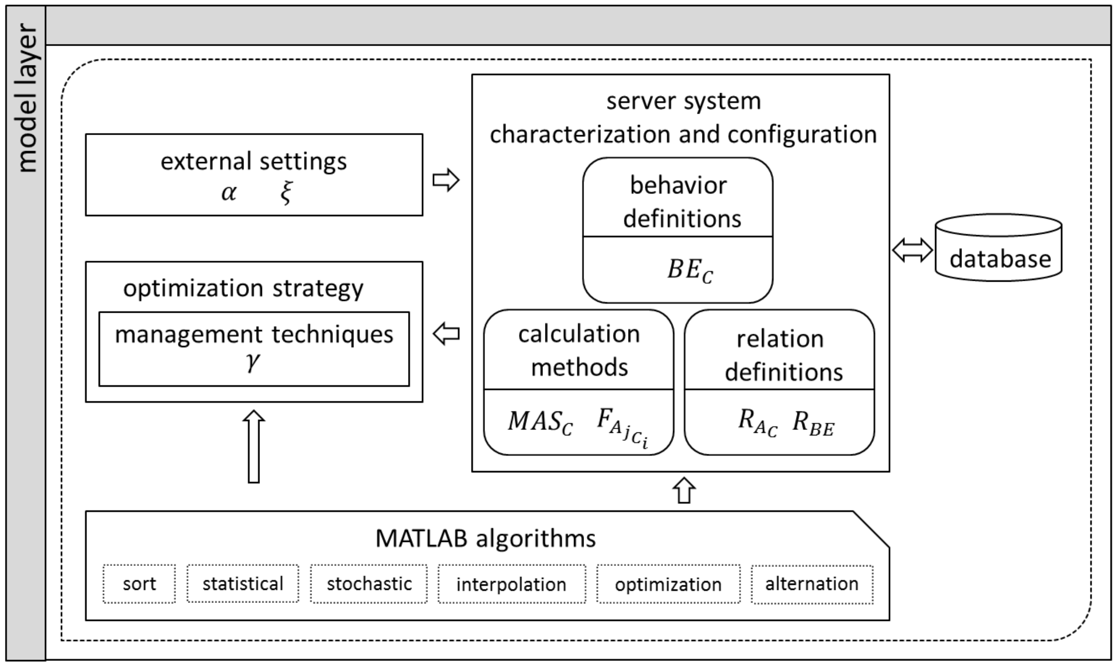


Figure 90: Model layer – block diagram

We implement the aspect-based calculation methods for the lookup-based models of every component. We specify the category-specific equations in the concept, as shown in Table 39.

Table 39: Calculation methods

| Category (component) | Power equations (section) | Temperature equations (section) |
|---|---|---|
| **Processor** | (5.76) - (5.75) (5.2.2) | (5.89) - (5.92) (5.2.2) |
| **Memory** | (5.66) - (5.68) (5.2.2) | (5.87), (5.88) (5.2.2) |
| **PSU** | (5.50) - (5.52) (5.2.1) | (5.87), (5.93), (5.94) (5.2.2) |
| **Input/output, others** | Static offset | Static offset |

For each category, we specifically define a method of *calc_category()*[301], which Simulink calls at each time step during the simulation, see Equation (6.1). We integrate the function call of *calc_category()* within the component models in Simulink[302] and specify the current state on the basis of the dynamic characteristics $CH_{CFG}^{DY}$, which we consider in MATLAB to be grouped as *settings*. We specify the descriptive *metadata*, which includes further details regarding the

[301] Names in MATLAB notation: we simplify the names to ensure a good readability, we use descriptive names in our methods or internal variables with longer text strings

[302] Component models in Simulink: see Section 6.3

characteristics in a separate database[303] to be unique and provide data consistency. We can integrate various dynamic and static characteristics in Simulink, but this requires a data conversion of every signal in each category providing them as numerical values[304]. We implement the aspect-based calculation in MATLAB because of lower development time and effort[305] in comparison to a Simulink-based approach. We include the ambient temperature and the category-specific utilization levels as the input signals *input_args* in the Simulink environment, which may change at each time step.

```
function [output_args] = calc_category(input_args)                (6.1)
  input_args = {input_args settings metadata}
  current_state = calc_state(input_args)
  input_args = {input_args current_state}
```

We specify the possible range of states in *calc_state()* and decide upon the current state on the basis of the utilization level, the read-to-write ratio, and the frequency in our memory-based method[306]. The range of states relies upon the *settings*, such as the memory generation, which we calculate considering the *metadata*. We have grouped the diverse memory states[307] into clusters considering the centroids by applying the simple k-means clustering algorithm [TSK 2009], which MATLAB provides in the stochastic toolbox as a cluster-based analysis method. We consider three major clusters, which we differentiate between high, medium, and low power consumption while executing various workloads. Figure 91 exemplarily shows the results of the k-means algorithm, which clusters the given set of $IDD$ values, as shown in the right-hand legend. We consider the centroids of $35mA$, $70mA$, and $155mA$, which correspond to a set of $IDD$ states. As a result of our analysis, we reduce the complexity of the $IDD$ states and define the baseline power[308].

[303] Database: separate m-files, dynamic structures, or structured information (matrix, vector)
[304] Numeric values in Simulink: support data types, such as single, double, signed, unsigned, 8/16/32-bit integer, Boolean, or fixed-point
[305] Development effort: adjusting code in MATLAB is easier than in Simulink, such as adding a signal or characteristic
[306] Category-specific states: memory (read-to-write ratio, frequency), processor (voltage-frequency pair)
[307] Memory states: memory currents $IDD$s, see Section 5.2.2
[308] Baseline power: voltage is a linear factor, power highly depends upon the current ($IDD$) states

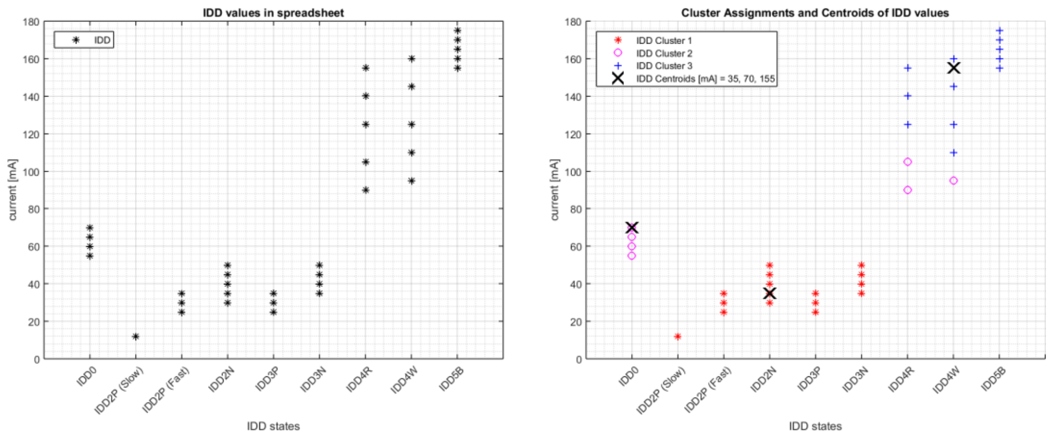


Figure 91: Clustering of $IDD$ memory states

In principle, each *calc_category()* method consists of the aspect-based subroutines *calc_aspect()* in the operating sequence of the power calculation, thermal calculation, and performance estimation. We define the component temperature on the basis of the power values, which thus becomes an additional input argument. The performance of a component depends upon the current temperature. We decouple the various methods and their databases to support a high flexibility when an adaption is necessary.

```
function [output_args] = calc_category(input_args)                    (6.2)
  [total_category_power] =
    calc_power(input_args)
  [total_category_temperature] =
    calc_temperature(input_args, total_category_power)
  [total_category_performance] =
    calc_performance(input_args, total_category_temperature)
```

We divide each aspect-based subroutine into three segments, considering the dynamic characteristics, the static characteristics, and the technical specification unique to a certain class, which we define in a list ordered by their significance. We exemplarily describe the power calculation method of a memory module ($PO_{mem}$), whereby the significant classes are *modes* ($CL_{mod}$), *technology* ($CL_{tec}$), and *manufacturing process* ($CL_{map}$), as defined in Equation (6.3). We show the memory-related characteristics in Table 26 and define the class-specific notation in Table 30.

$$PO_{mem} = \begin{cases} CL_{mod} \rightarrow corr, volt, frequency \\ CL_{tec} \rightarrow cap, dens, sync, ranks, ranks\_x \\ CL_{map} \rightarrow type, arch, gen, fam, series, vendor, plc, die \end{cases} \tag{6.3}$$

On the basis of the results of our analysis, we implement the *error correction* $(corr)$, *synchronization mode* $(sync)$, and *die* technology as independent linear factors in our memory power calculation method. We specify the dynamic structures[309] by a header[310] that specifies the characteristic naming, provides the direction, and defines the kind of dependencies $(dep)$ between the values $(val\_1, val\_2, val\_3)$, such as an absolute or relative factor. In our example, we define the linear factor[311] $(0.9)$ when the value of the characteristic changes of $val\_1$ towards $val\_2$, as shown in Equation (6.4). We can easily adjust the dynamic structure by adding a new value or update the dependencies.

```
characteristic=[{'header'} {'val_1'} {'val_2'} {'val_3'}];      (6.4)
 dimension = length(characteristic);
 % factor between values [0 0.9 1.3 0];
 dep(1:dimension) = 0;
 dep(2) = 0.9; dep(3) = 1.3;

for i = 1:dimension                                              (6.5)
 characteristic_all(i).id = characteristic(i);
 characteristic_all(i).value = dep(i);
end
```

In contrast, the characteristics of the technology class $(characteristic\_tec)$, such as *capacity* $(cap)$, *density* $(dens)$, *rank* $(ranks)$, and *rank linking* $(ranks\_x)$, rely upon each other, see Section 5.2.2. Therefore, we define a system of linear equations in a matrix to specify the interdependencies, as shown in Equation (6.6). The technology factor $tec\_factor$ is the summation of the certain dependencies $dep$, which we adjust by the $corr$ factors of the different memory modules and settings, see Equation (6.7).

```
% characteristic_tec [ ranks capacity density ranks_x ]          (6.6)
% [ ranks[%]; capacity[%]; density[%]; ranks_x[%] ]
characteristic_tec = [1 1 1 1; 2 1 2 1; 2 1 1 2; 1 1 2 0.5];
rel_dep = [0; 0.13; -0.32; -0.64];
dep = characteristic_tec/rel_dep;

tec_factor = dep(1)*ranks*correct.ranks +                        (6.7)
dep(2)*cap*correct.cap + dep(3)*dens*correct.dens +
dep(4)*ranks_x*correct.ranks_x;
```

In our optimization phase, we require the current customer-specific configuration $\theta_C$ and the configuration trees $\theta, \theta_{TS}, \theta_{CS}$, which we can only handle in MATLAB[312]. We consider the initial configuration $\theta_C$ and its base characteristics to set the default settings of each calculation

[309] Dynamic structures: using vector- and matrix-based dimensions optimized in MATLAB
[310] Header of a dynamic structure: {'identifier'} {''} {'basic calculating operation'} {'direction'}, basic calculating operation (*,/,-,+), direction (upwards vs. downwards), absolute vs. relative
[311] Factor: a factor equal to zero does not have any dependency on the next value in the direction
[312] Input arguments in Simulink: require numeric values as signals between the blocks

method. We define the possible range of characteristics in a separate database and specify the weight coefficients $WF^{CH}_{A_{jC_i}}, WF^{CL}_{A_{jC_i}}, WF_{TS}, WF^{ST}_{CFG}, WF^{DY}_{CFG}$, which we consider in the *metadata.* Finally, we consider the class-specific factors and weight coefficients in the power calculation method of a memory module.

$$PO_{mem} = f(CL_{mod}, CL_{tec}, CL_{map}) \tag{6.8}$$

$$CH^{DY}_{CFG} = \{corr, volt, frequency, utilization\ level, read - to - write\ ratio\} \tag{6.9}$$

$$CH^{ST}_{CFG} = \{cap, dens, sync, ranks, ranks\_x\} \tag{6.10}$$

$$CH_{TS} = \{type, arch, gen, fam, series, vendor, plc, die\ \} \tag{6.11}$$

Figure 92 briefly presents the internal workflow of the calculation methods.

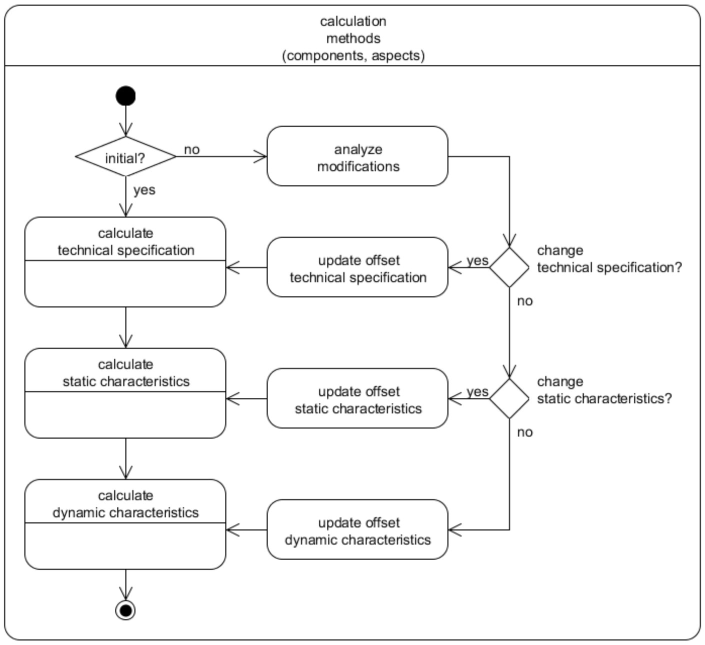

calculation methods (components, aspects)
initial?
no
yes
analyze modifications
calculate technical specification
update offset technical specification
yes
change technical specification?
no
calculate static characteristics
update offset static characteristics
yes
change static characteristics?
no
calculate dynamic characteristics
update offset dynamic characteristics

Figure 92: Workflow – calculation methods

The main outputs *output_args* of the *calc_category()* methods are the results of each aspect, which influence each other. Thus, we exemplarily specify the relation definition $R_A$ between the power and temperature of the memory module, whereby we consider the formal definition of Equation (5.93). We specify the static memory temperature, including the memory power consumption, the weight coefficient $WF^{TH}_{mem}$ replaced by the thermal resistance $\theta_{res}$, and the ambient temperature $\alpha_{temp}$ as the thermal offset. We define the dynamic thermal development on the basis of the empirical measurements gained from a real server system, as shown in Figure 93. We apply the *lsqcurvefit* method of MATLAB to approximate the measured temperatures.

We define the first-order differential equation that best suits to describe the dynamic thermal development of a memory module. In Equation (6.13) we specify the PT1 method of the coefficients $K_{mem}^{ST} = 13.07$, $T_S = 87.33s$, and $TH_{mem}^{ST} = 33°C$, which result from the approximation.

$$TH_{mem}^{ST} = PO_{mem} * \theta_{res} + \alpha_{temp} \tag{6.12}$$

$$TH(t)_{mem}^{DE} = 13.07 * \left(1 - e^{-\frac{t}{87.33}}\right) + 33 \tag{6.13}$$

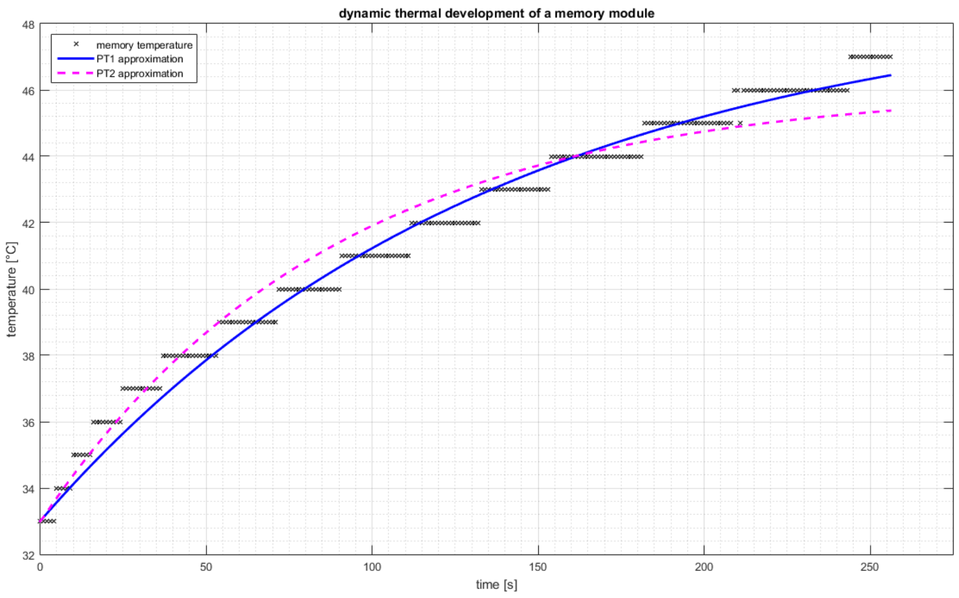


**Figure 93: Dynamic thermal development of a memory module**

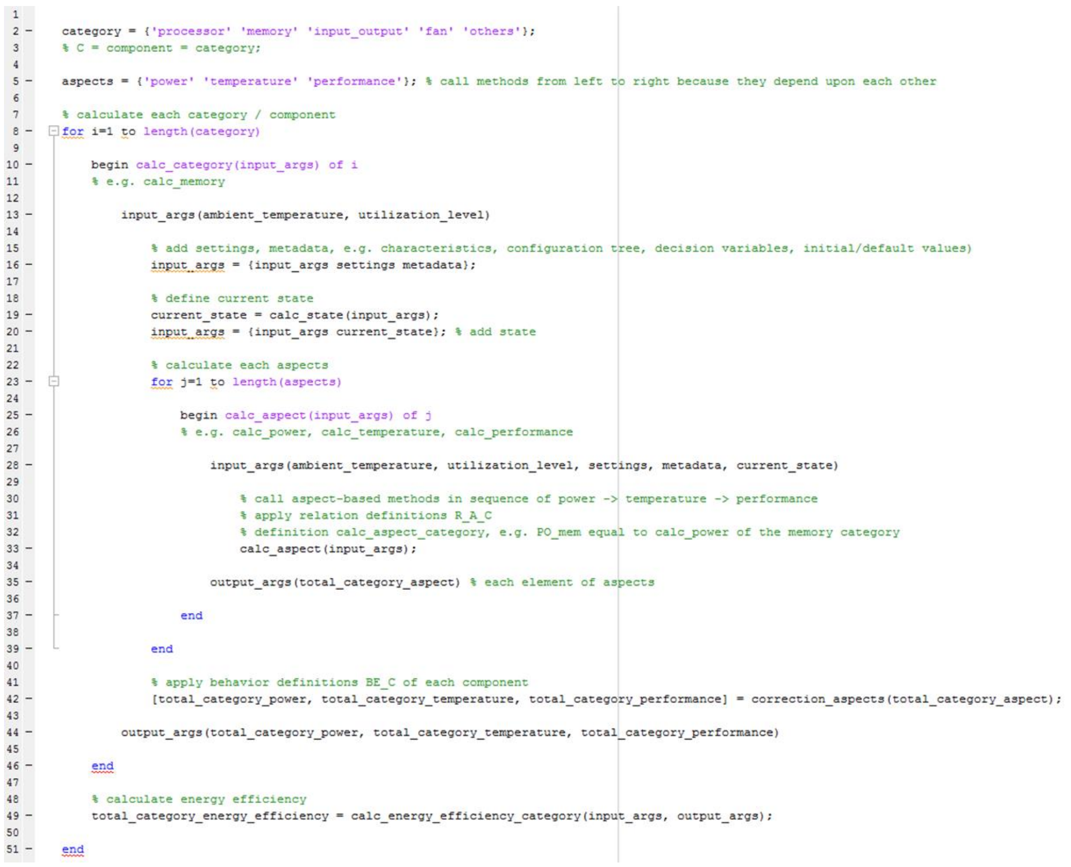
We consider the thermal interaction between various components of the same category as a part of the relation definition $R_A$. A small mutual distance between the memory modules leads to a higher thermal impact. The processors have a large distance and active cooling and therefore smaller temperature impacts on each other. The processor's thermal dissipation in an enclosure significantly depends upon the surrounding air temperature itself, as stated in [KLL et al. 2008]. The authors of [LZZ et al. 2007] neglect the thermal effects in their approach and consider the components in isolation.

We simply map the memory utilization levels considering the workload scenario of the memory performance, which we store in a database. The performance scaling is an important constituent of the relation definition $R_{BE}$ when we alternate the components. We include the strategy addressed in [DEP et al. 2009], which specifies the performance improvements because of some adjustments at the processor and memory configuration. Table 40 exemplarily shows the coefficients when we double the number of processors and memory capacity at the same time. We consider the approximate coefficients from 1.5 up to 1.7 to estimate the performance of the system.

Table 40: Performance improvement (processor and memory capacity) [DEP et al. 2009]

| Step | Processor amount and memory capacity | Performance improvement (coefficient) from step *n* to *n+1* |
|---|---|---|
| **1** | 1 CPU 4 GB | 1.7 |
| **2** | 2 CPU 8 GB | 1.6 |
| **3** | 4 CPU 16 GB | 1.5 |
| **4** | 8 CPU 32 GB | 1.5 |
| **5** | 16 CPU 64 GB | unknown |

Figure 94 shows the pseudo code of the methods and categories that we implement in MATLAB. We implement all categories and respective component methods in MATLAB with nominally identical parts presenting unique values, which follow the same principles as used when we describe the memory module. We implement the behavior between the components $R_{BE}$ within the Simulink model. We strictly follow our concept concerning the optimization strategy and alternation, which we do not explain in further detail.



```

category = {'processor' 'memory' 'input_output' 'fan' 'others'};
% C = component = category;

aspects = {'power' 'temperature' 'performance'}; % call methods from left to right because they depend upon each other

% calculate each category / component
for i=1 to length(category)

    begin calc_category(input_args) of i
    % e.g. calc_memory

        input_args(ambient_temperature, utilization_level)

            % add settings, metadata, e.g. characteristics, configuration tree, decision variables, initial/default values)
            input_args = {input_args settings metadata};

            % define current state
            current_state = calc_state(input_args);
            input_args = {input_args current_state}; % add state

            % calculate each aspects
            for j=1 to length(aspects)

                begin calc_aspect(input_args) of j
                % e.g. calc_power, calc_temperature, calc_performance

                    input_args(ambient_temperature, utilization_level, settings, metadata, current_state)

                        % call aspect-based methods in sequence of power -> temperature -> performance
                        % apply relation definitions R_A_C
                        % definition calc_aspect_category, e.g. PO_mem equal to calc_power of the memory category
                        calc_aspect(input_args);

                    output_args(total_category_aspect) % each element of aspects

                end

            end

            % apply behavior definitions BE_C of each component
            [total_category_power, total_category_temperature, total_category_performance] = correction_aspects(total_category_aspect);

        output_args(total_category_power, total_category_temperature, total_category_performance)

    end

    % calculate energy efficiency
    total_category_energy_efficiency = calc_energy_efficiency_category(input_args, output_args);

end
```

Figure 94: Pseudo code – aspect-based and category-based calculation process

The user of our simulation framework should reckon with some slight changes in the implementation in comparison to the concept of higher efficiency. We implement the Simulink-based component models considering the *calc_category()* methods, which constitutes the highest abstraction level. The internal functions such as the aspect-based subroutines *calc_aspect()* represent the diverse hierarchy levels of our concept, wherein we consider the various classes, the related dynamic and static characteristics, or the technical specifications. We provide modularity because we can replace or extend a single calculation method considering various characteristics dependent on the actual empirical results. Our simulation framework consists of approximately 18,500 physical lines of code (LOC) distributed over 250 files, including almost 33 percent comments. We implement around 220 m-scripts with nearly 260 functions and around 30 Simulink models (*.slx).

**Restricting Assumptions in Order to Simplify the Calculation**

We consider several components that can fulfill the requirements of a certain workload scenario, which provides the opportunity to influence the energy consumption. Theoretically, we can consider all server configurations and characteristics, but we exemplarily concentrate upon a couple of server systems from which we can easily receive experimental results (measurements, benchmarks). We restrict our prototype implementation to demonstrate the feasibility of our concept by reducing the complexity to decrease the calculation effort and simulation time. We consider the aspect-based models of the selected system configuration $\theta_C$ and its components, which we limit to rack-based server systems, and fix the component behavior as a simplified assumption. We assume that the concept is applicable to blade and tower servers with slight adoptions concerning their architecture. We manually implement the system architecture and connectors in Simulink, which are temporary static in our simulation framework. We neglect the operating systems influences, as described in Section 5.3.2.3. We assume a constant ambient temperature of the server system, while executing the workload scenario that is based upon the regular functionality of the heating, ventilation, and air conditioning (HVAC) equipment in the data center. We support the ambient temperature range of our GUI as shown in Table 41.

Table 41: GUI-based restrictions of the server system

| Parameters | Range |
|---|---|
| Ambient temperature [°C] | {'20°C - 25°C'} {'25°C - 30°C'} {'30°C - 35°C'} {'35°C - 40°C'} {'40°C - 45°C'} |
| Workload scenario | {'Steady'} {'Continuously'} |
| Workload profile | {'Idle'} {'Transactional'} {'Computational'} {'Memory-intensive'} {'Worst-case'} |
| Utilization levels [%] | [0:1:100] |
| Platform segment | Single rack-based server system |
| Server system model | Fujitsu RX200, RX300 family |
| Observable subsystems | Processor, memory, fan, PSU |
| Quantity (#) processors | 1 - 2 |
| # memory modules | 1 - 24 |
| # fans | 1 - 16 (enclosure) |
| # power supply units | 1 - 2 |

We restrict the feasible range of controllable variables, such as the decision variables, on the basis their relevance in our category-specific and aspect-based calculation methods without any claim of comprehensiveness. Table 42 and Table 43 show the restrictions of the memory characteristics.

Table 42: Memory characteristics – simulation parameters ($I$)

| Memory characteristics | Range |
|---|---|
| Quantity (#) | 1 - 64 |
| Vendor | {'Micron'} {'Samsung'} {'Hynix'} {'Qimonda'} {'Netlist'} |
| Capacitors / capacitance capacity (size) [GB] | {'1GB'} {'2GB'} {'4GB'} {'8GB'} {'16GB'} {'32GB'} |
| Generation | {'DDR3'} {'SDRAM DIMM'} |
| Family | 240 Pin DDR3 DIMM |
| Series | Family & synchronization mode & capacity & die |
| Density [GB] | {'1GB'} {'2GB'} {'4GB'} {'8GB'} {'16GB'} {'32GB'} |
| Die (component revision) | {'M'} {'G'} {'F'} {'E'} {'D'} {'C'} {'B'} {'A'} |
| Fabrication size [nm] | {'56nm'} {'54nm'} {'46nm'} {'44nm'} {'38nm'} {'35nm'} {'29nm'} |
| Synchronization mode | {'load reduced'} {'registered'} {'unbuffered'} {'fully buffered'}; {'LR'} {'R'} {'U'} {'FB'} |

Table 43: Memory characteristics – simulation parameters $(II)$

| Memory characteristics | Range |
|---|---|
| Module ranks, rank linking (data width) | {'1R'} {'2R'} {'4R'}; {'SR'} {'DR'} {'QR'} {'x4'} {'x8'} {'x16'} |
| Timings | {'6'} {'7'} {'8'} {'9'} {'10'} {'11'} {'13'} |
| Resistance | '56°C/W' |
| Interleaving | {'bank'} {'channel'} {''} |
| Error correction | {'ECC'} {''} |
| Refresh | {''} {'mirroring'} {'independent'} {'sparing'} {'scrubbing'} |
| Frequency [MHz] | {'400'} {'533'} {'667'} {'800'} {'933'} {'1066'} |
| Voltage [VDC][313] | {'LV'} {'STD'}; 1.35 - 1.5VDC |
| Transfer rate / throughput [MHz] | {'800'} {'1066'} {'1333'} {'1600'} {'1866'} {'2133'}, {'PC3-6400'} {'PC3-8500'} {'PC3-10600'} {'PC3-12800'} {'PC3-14900'} {'PC3-17066'} |

We consider the processor characteristics, which we restrict to a single processor family. We exemplarily describe our results of the *Intel Xeon* architecture – the third generation, code name *Ivy Bridge*. We simulate the *E5-2600* product family of the *E5-v2* processor family. We implement the aspect-based calculation methods to consider, especially the *Intel Xeon E5-2690v2* and *Intel Xeon E5-2670v2* processors. We analyze the power gap between the spreadsheet-based estimating, the measurements, and our simulation results. Table 44 and Table 45 show the processor characteristics, which we exemplarily support in our prototype implementation.

Table 44: Processor characteristics – simulation parameters $(I)$

| Processor characteristics | Range |
|---|---|
| Cache / cache lines [MB] | {'10MB'} {'15MB'} {'20MB'} {'25MB'} {'30MB'} |
| Voltage [V] | 0.65-1.3V |
| Time | Consider various device states in time intervals |
| Semiconductor technology (TDP) [W] | {'60W'} {'70W'} {'80W'} {'95W'} {'115W'} {'130W'} |
| Quantity (#) | 1 - 4 |

[313] VDC: volts direct current

Table 45: Processor characteristics – simulation parameters $(II)$

| Processor characteristics | Range |
|---|---|
| **Status** | Enabled, disabled |
| **Type (OS)** | Microsoft Windows Server 2012R2, manufacturing process |
| **Vendor** | {'Intel'} |
| **Product life cycle stage** | Market introduction, growth, maturity, saturation, and decline |
| **Architecture** | *Intel Xeon E5* |
| **Generation** | Ivy Bridge EP |
| **Family** | *E5-2600v2* |
| **Series** | {'Intel Xeon E5-2603v2'} {'Intel Xeon E5-2680v2'} {'Intel Xeon E5-2670v2'} {'Intel Xeon E5-2660v2'} {'Intel Xeon E5-2650v2'} {'Intel Xeon E5-2640v2'} {'Intel Xeon E5-2630v2'} {'Intel Xeon E5-2620v2'} {'Intel Xeon E5-2609v2'} {'Intel Xeon E5-2690v2'} {'Intel Xeon E5-2697v2'} {'Intel Xeon E5-2695v2'} {'Intel Xeon E5-2667v2'} {'Intel Xeon E5-2643v2'} {'Intel Xeon E5-2637v2'} {'Intel Xeon E5-2650Lv2'} {'Intel Xeon E5-2630Lv2'} |
| **Fabrication size [nm]** | {'22nm'} |
| **Resistance** | '0.257°C/W' |
| **Performance features (turbo)** | {'yes'} {'no'}, *Intel VT-x, AMD-V* |
| **Cores / active cores (hyper-threading) [C,T]** | {'1C'} {'2C'} {'3C'} {'4C'} {'5C'} {'6C'} {'7C'} {'8C'} {'9C'} {'10C'} {'11C'} {'12C'} {'13C'} {'14C'} {'15C'} {'16C'}, {'1T'} {'2T'} {'4T'} {'8T'} {'12T'} {'16T'} {'20T'} {'24T'} |
| **Frequency [GHz]** | {'1.2'} {'1.3'} {'1.4'} {'1.5'} {'1.6'} {'1.7'} {'1.8'} {'1.9'} {'2.0'} {'2.1'} {'2.2'} {'2.3'} {'2.4'} {'2.5'} {'2.6'} {'2.7'} {'2.8'} {'2.9'} {'3.0'} {'3.1'} {'3.2'} {'3.3'} {'3.4'} {'3.5'} {'3.6'} {'3.7'} {'3.8'} |
| **Accesses / instructions / operands** | Integer, floating-point |
| **Transfer rate [GT/s, MHz]** | {'6.4GT/s'} {'7.2GT/s'} {'8.0GT/s'}, {'1333MHz'} {'1600MHz'} {'1866MHz'} |
| **Thresholds (thermal)** | 63°C - 88°C |

We statically characterize the fan-specific behavior, which specifies how rapidly we will absorb and dissipate the heat. Our fan model is based on the characteristics presented in Table 46.

**Table 46: Fan characteristics – simulation parameters**

| Fan characteristics | Range |
|---|---|
| Current [A] | 0.02A - 2.5A |
| Speed [RPM] | 0 – 20000 |
| Type | Axial-flow, centrifugal |
| Voltage [VDC] | 10.8 - 12.6VDC |

The model layer provides the mathematical methods and configuration data that we require in the simulation. The Simulink environment calls the functions receiving the energy efficiency of each component, which we control and monitor.

## 6.3 Controller Layer

The philosophy of Simulink considers a monolithic controller in the block diagram, an approach that we cannot apply concerning our concept because we adjust the dynamic behavior of each component. The controller layer of our simulation framework generally consists of a controller template according to Simulink (denoted as Simulink controller) and a set of distributed controllers that are specific to each component (named as component controller). The Simulink controller manages the simulation instructions provided by the view layer and specifies several simulation options, such as the initial conditions, the sample time of the Simulink blocks, or the start and stop time of a simulation run. We define the external interfaces for the simulation model in the Simulink controller and create the input signals by the *stimuli model*, which loads the utilization levels from the GUI and assigns them to our system model in Simulink. We connect the component models by signals in Simulink to enable the communication in the Simulink controller, which distributes and shares the aspect-based signals between each other. We specify the output signals to store the simulation results, which the Simulink controller manages.

We design the top-level structure and architecture of a rack-based server system in Simulink, which includes the lookup-based models of each component, as specified in Section 6.2. We implement each component as a stand-alone Simulink model that supports the independent specification of the internal component behavior. The Simulink controller calls the aspect-based calculation methods, which we integrate into the component models. We can exchange or simply adjust the calculation methods in MATLAB without changing the simulation model itself. We arrange the models in a modular approach, enabling them to communicate in a loosely coupled fashion to ensure scalability and exchangeability.

We split the expected monolithic controller of the Simulink model into several component controllers to enable especially the primary phase, wherein we customize the dynamic behavior. An exception is the *fan model*, which includes an internal monolithic controller (fan speed control) and considers the thermal development of the entire server system. Consequently, we connect the *fan model* to all components in our Simulink controller. The *power supply* is a non-controllable component that provides the power on the basis of the PSU efficiency and redundancy settings.

We *visualize and monitor* the results of the *system model* at each time step of the simulation in the Simulink controller. Afterwards, we *analyze and evaluate* the impact of the diverse server configurations and characteristics on the energy efficiency ratio under various workloads and vendor-specific constraints. We decide upon the alternation strategy in our *system-wide optimization engine*, which results in an adjusted *system model* and requires updates on each instance of the distributed component controller. Figure 95 shows the simplified structure of the controller layer as a block diagram.

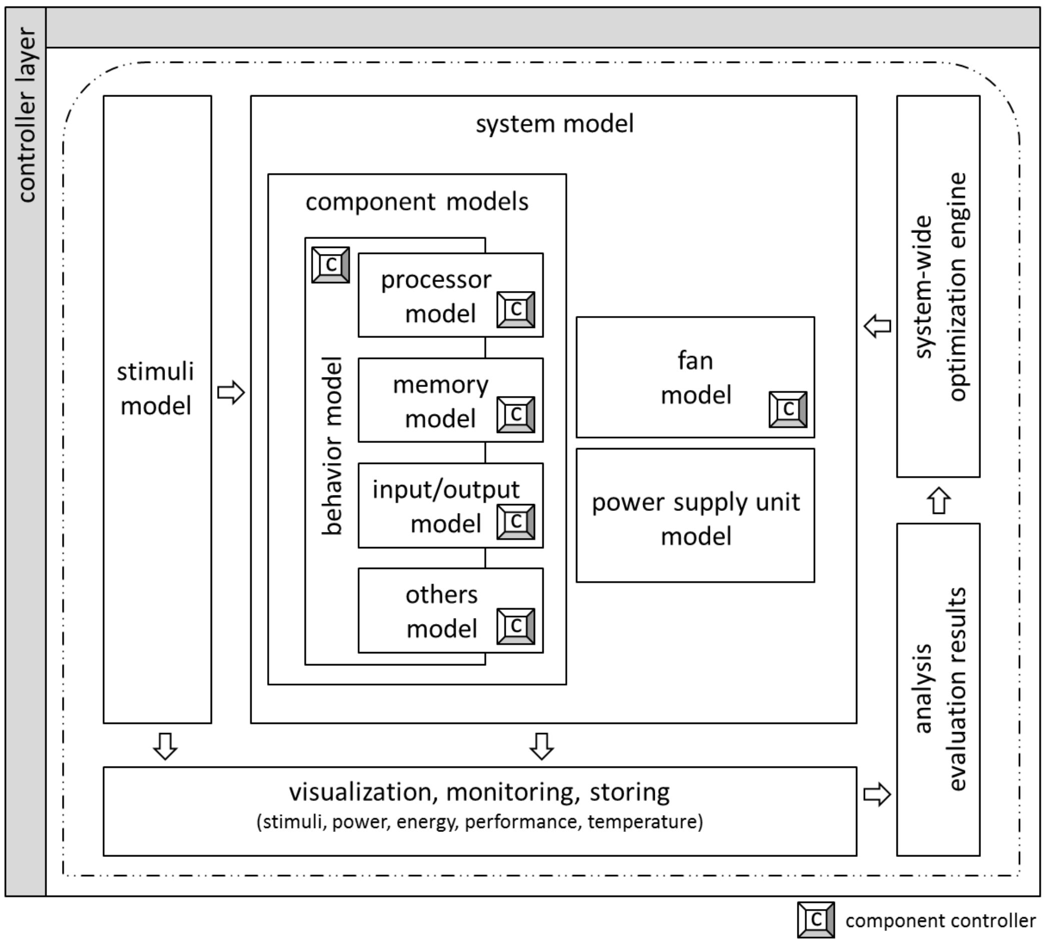


Figure 95: Controller layer – block diagram

Our simulation model consists of two major blocks at the highest abstraction level, as shown in Figure 96. The *stimuli* model[314] generates the customer-specific (steady or continuously) utilization levels of every component. We consider the ambient temperature of the rack-mounted server system as a static input. We visualize the simulation results in the Simulink scopes[315] to enable the update process between the MVC layers.

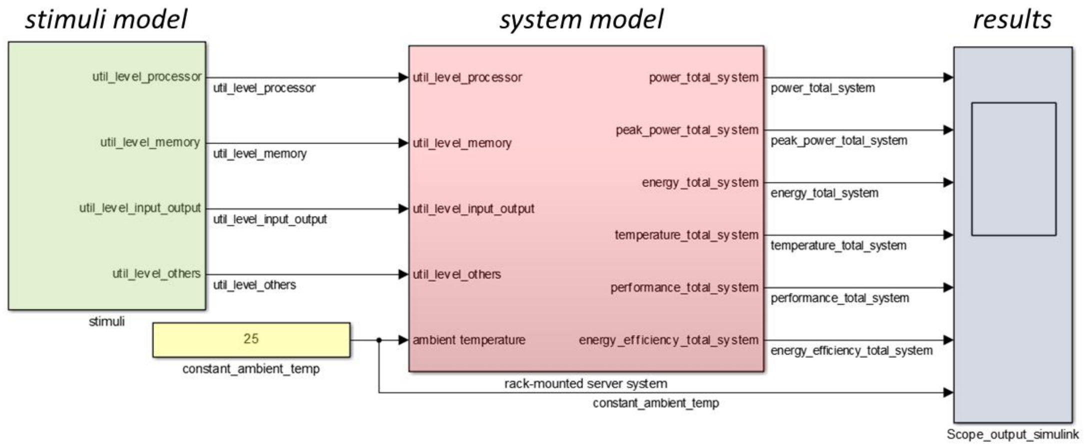


Figure 96: Simulation model – stimuli and server system

Figure 97 shows the simplified subsystems of the rack-mounted server system, including the components, thermal control, and the power supply unit, which we divide into several subsystems. The *components* subsystem contains separate subsystems of the *processor*, *memory*, *input/output*, and *others* categories, which behave as individuals specified by the mathematical methods described in Section 6.2. Each component provides the performance, power, and thermal values that we couple with each other to provide the resulting values. The thermal control influences the temperatures inside the system, which we consider as the same all around the enclosure. We implement the thermal control, according to the specification in the concept chapter. In general, we implement a fan analogue to a component, but it does not include the performance calculation. We sum up the power consumption values resulting in the secondary power (provided by the PSU) and calculate the primary power (input to the PSU) considering the power supply efficiency and redundancy settings. We want to refrain from a detailed description of the subsystems[316], because of the reduced readability in the figures.

[314] Stimuli model: Simulink subsystem, implement in the *workload generator* in MATLAB
[315] Simulink scope: define a name tag
[316] Subsystems: the customer or user can get further details by double-clicking at the Simulink blocks

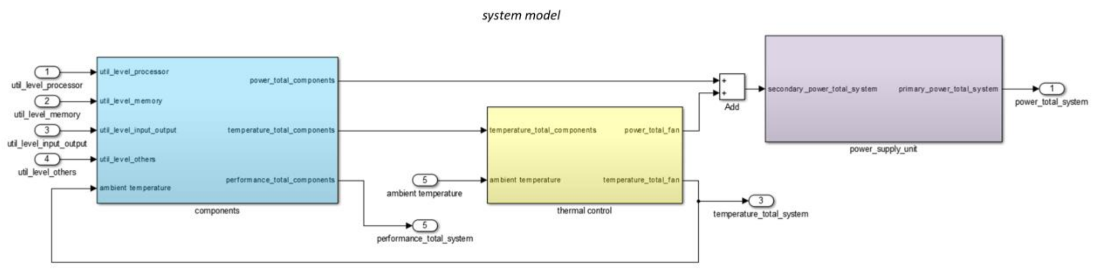


**Figure 97: System model – components, thermal control, and power supply unit (simplified representation)**

Hence, we implement the MATLAB methods and Simulink models; we execute the MATLAB script *GUI.m*, which builds the simulation entry and initializes the graphical user elements by their related functions. We load the configuration file of the server system and parse it to adjust our configuration tree and restrict the decision variables. Afterwards, we set up the model and simulation parameter, which we require to control the Simulink model in the background. The graphical user interface remains at the wait state as long as an event occurs, see Figure 98. The MATLAB callback functions react to any changes in the graphical user interface, such as the workload scenario (*popup menu*), the utilization levels (*sliders*), the selection of the server configuration (*list box*), or the start request (*button*). We implement the specific callback functions and provide the settings to the framework[317], which parameterizes the simulation model. We activate the simulation and optimization by pressing the start button in the GUI.

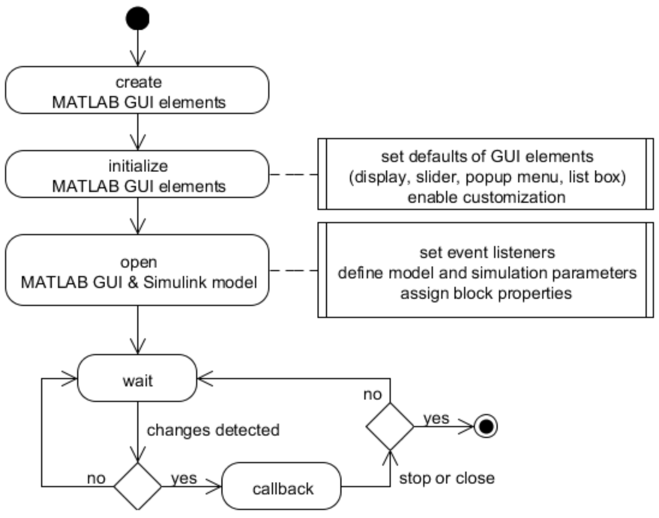


**Figure 98: Workflow – MATLAB GUI and Simulink model**

[317] Framework: MATLAB scripts (*.m files), MATLAB workspace, and Simulink (model workspace, block properties)

We realize our framework regarding the concept, which includes both a pre- and post-process of the proper simulation, as shown in Figure 99.

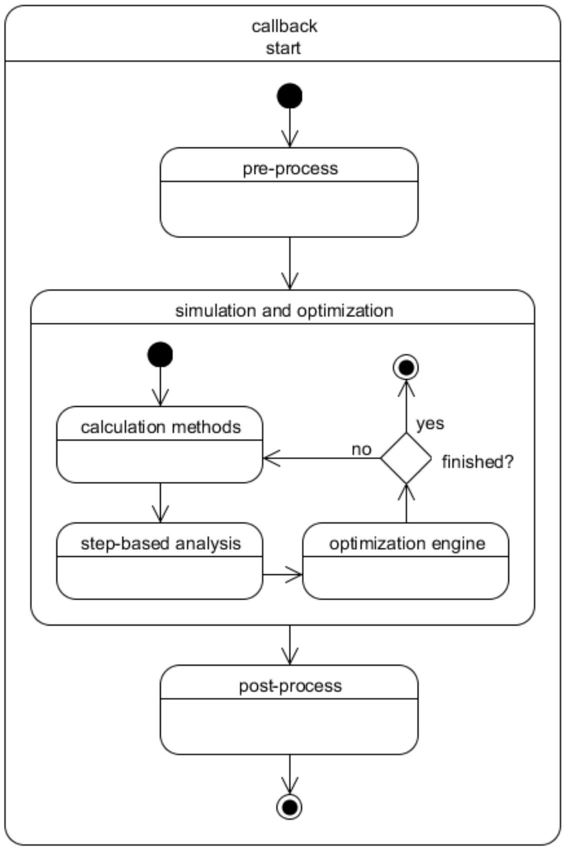


Figure 99: Workflow – callback simulation start

In the pre-process, we import the workload scenario and pre-define the stimuli when we open the GUI, as described in Section 6.1. We update the stimuli on the basis of any change in the GUI. In parallel, we create the generic configuration tree and import the customer-specific server configuration, which restricts the system-compatible hardware and the related decision variables. We initialize the aspect-based methods, as we briefly explain the procedure in Section 6.2. Figure 100 shows a simplified workflow of the pre-process.

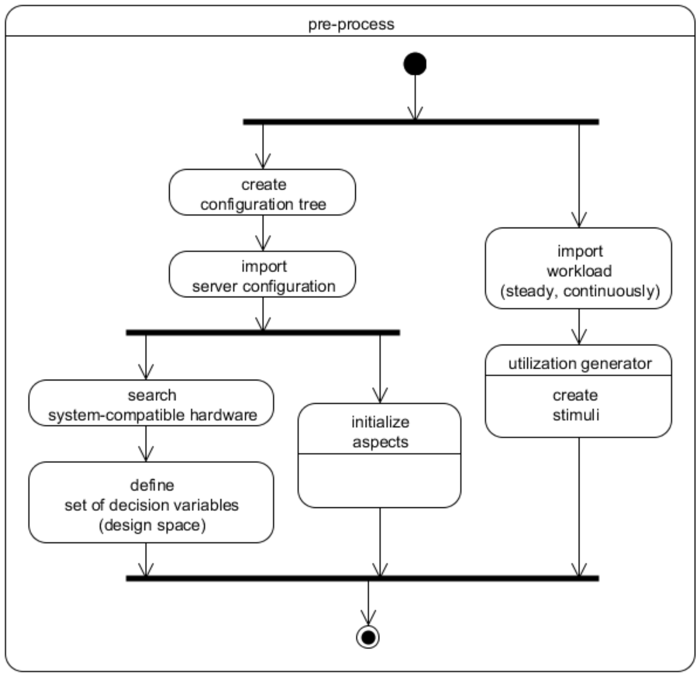

pre-process
create configuration tree
import server configuration
search system-compatible hardware
define set of decision variables (design space)
initialize aspects
import workload (steady, continuously)
utilization generator
create stimuli

Figure 100: Workflow – simulation pre-process

In each calculation method, we identify the relevant aspect-based characteristics of the respective component and load them into the simulation environment. Afterwards, we establish the particular weight coefficients and apply the offset of our initial configuration in relation to the default values in our database. We call the calculation methods and create the lookup-based models. Figure 101 presents an overview of the initialization workflow.

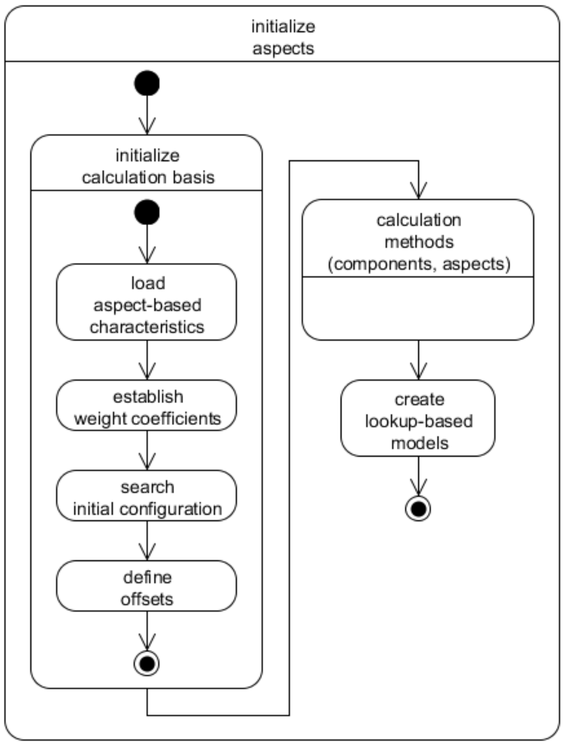

initialize aspects
initialize calculation basis
load aspect-based characteristics
establish weight coefficients
search initial configuration
define offsets
calculation methods (components, aspects)
create lookup-based models

Figure 101: Workflow – initialize aspect-based methods

After configuring and initializing the calculation methods, we execute the simulation and optimization. Figure 102 shows the workflow of the simulation framework in an abstract manner. Our simulation framework considers the utilization levels at each time step $t_k$, which controls the simulation loop. We synchronize the time stamps of the workload scenario of the timer of the simulation model. In the first optimization loop, we consider the initial set of decision variables in our calculation method, which consists of the lookup-based models. We load the utilization levels at the time $t_k$ and calculate the aspects of the energy efficiency computation that we consider. We strictly follow our concept concerning the step-based analysis, presented in Figure 79. We evaluate the results and find the possible impacts of the several characteristics on whose basis we decide on the alternation strategy. We restrict the set of decision variables that we further want to modify, see Section 5.4.2.2. If we modify a characteristic[318], we update the lookup-based models and call the calculation methods again. This results in a recursive optimization at each time $t_k$. If we do not require a subsequent iteration within the optimization loop, we define the step-based optimum and advance to the next time step $t_{k+1}$, as long as we do not reach the end of the workload scenario. We continuously visualize the aspects in the Simulink scopes and save the values in our environment. We globally analyze the step-based optima in the post-process, which we implement in MATLAB.

[318] Modification of the characteristics: change the values

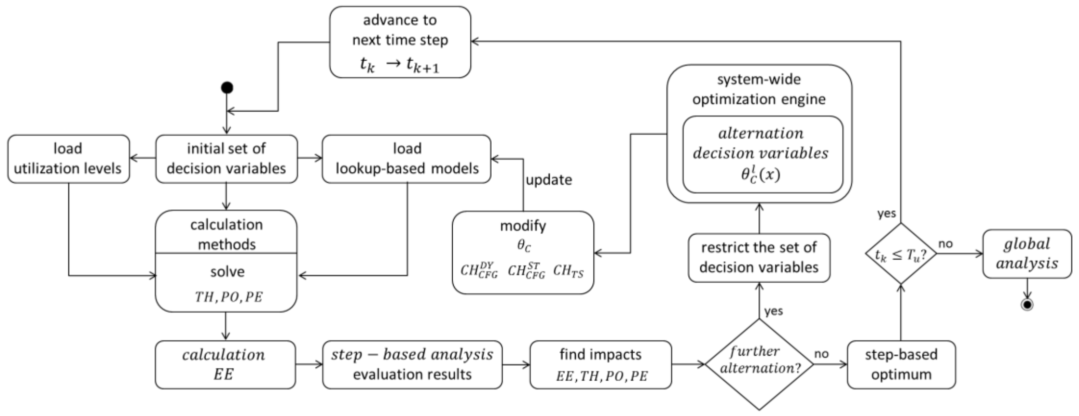

advance to next time step
$t_k \rightarrow t_{k+1}$
load utilization levels
initial set of decision variables
load lookup-based models
update
calculation methods
solve
$TH, PO, PE$
modify
$\theta_C$
$CH_{CFG}^{DY}$ $CH_{CFG}^{ST}$ $CH_{TS}$
system-wide optimization engine
$alternation$ $decision\ variables$ $\theta_C^l(x)$
restrict the set of decision variables
yes
$t_k \leq T_u?$
no
$global\ analysis$
$calculation\ EE$
$step - based\ analysis$ evaluation results
find impacts $EE, TH, PO, PE$
$further\ alternation?$
step-based optimum

Figure 102: Workflow of the simulation framework

## 6.4 Summary

We realize our concept as a Model-View-Controller (MVC) approach that considers a flexible amount of rack-based server configurations and components. The customer can define a workload scenario that specifies the specific demands in a more realistic manner. Furthermore, we integrate the ambient temperature of the server system, which reflects the data center requirements. We implement our framework as a combination of MATLAB and Simulink to gain each benefit, such as the existing realization of certain algorithms. In the next chapter, we evaluate our calculation methods of the components considering the energy efficiency of the entire server system.

# 7 Evaluation of the Multi-aspect Full-system Server Model and Optimization (MFSMOS)

In the first section, we introduce the evaluation environment in which we describe the measured server system, the software-based measurement infrastructure, and the benchmarks considering different server configurations. The aim of the evaluation is to prove the applicability of our developed concept as a proof-of-concept. We demonstrate the viability of our multi-aspect-based calculation methods for the components, which we integrate into a full-system server model. We emphasize the reliability, portability, and flexibility of our simulation framework and demonstrate that we are able to optimize a server system concerning its configuration and characteristics. To validate our models, we performed a sequence of benchmarks on various server system configurations and varied the utilization levels. To demonstrate the industrial feasibility and suitability, we emphasize the modular description of the workload and the server system configuration. We evaluate our *MFSMOS* approach considering a series of analyses with our prototype implementation and we assume the applicability of each alternative server system, considering another vendor or platform segment, for instance.

First, we evaluate the accuracy of the aspect-based calculation methods to check the plausibility, and then we analyze the impacts on particular characteristics to test the flexible reaction to possible changes. Here, we characterize the components and analyze the response of the varying utilization levels and consider a set of micro-benchmarks in order to distinguish the different component activities and consider the behavior in the entire system. Thirdly, we consider the worst-case power and energy efficiency in a case study in two versions, one without optimizations and one with. This is in accordance to common industry practice. Here, we present an experimental analysis to demonstrate the optimization possibilities and concentrate upon the three main analyses: *Accuracy Analysis*, *Impact on Characteristics Changes*, and *Energy Efficiency Analysis*.

## 7.1 Evaluation Environment

### 7.1.1 System under Test (SUT)

We analyze an exemplary rack-based server system, the system under test (SUT), which is a Fujitsu[319] server system. Table 47 shows a brief overview of the hardware settings, which allows up to two processors and 24 memory modules. Our current SUT is equipped with two *Intel Xeon E5-2650v2* processors, whereby we disable the second processor in the BIOS/UEFI[320] when we want to analyze a single processor in the system. We have access to up to 12 memory modules for test purposes. If we require fewer modules in our experiments, we physically remove the modules. We analyze the total memory capacity and related

[319] Fujitsu server systems: http://www.fujitsu.com/fts/products/computing/servers/primergy/rack/

[320] Disable a processor: It automatically disables memory modules that correspond to the processor.

characteristics in order to show the differences between the various benchmarks, for instance. The particular system's power supply delivers an output of $450W$ at around $94\%$ efficiency. In our experimental setup, we share one hard disk with the operating system, measurement tools, and benchmarks. The combination of the OS and the tools on a single hard disk may influence the evaluation results[321], such as throughput, response time, or latency.

Table 47: System under test (SUT) – hardware settings

| Category | Settings |
|---|---|
| **Platform segment** | Single rack-based server system |
| **Server system model** | Fujitsu PRIMERGY RX200 family (RX200S8) |
| **Form factor** | Dual socket 2U rack server |
| **Processor** | |
| **Family (Series)** $(C18)$ | *Intel Xeon E5-2600 v2 (E5-2650v2)* |
| **Generation** | Ivy Bridge EP (Romley) |
| **Frequency** | 1.8GHz – 2.1GHz, turbo 2.3GHz |
| **Hyper-threading / turbo** | Enabled / enabled |
| **Enabled** | 8 cores, 2 chips |
| **Hardware threads** | 16 (2 / core) |
| **L1 Cache** | 8x32KB instruction caches, 8x32KB data caches |
| **L2 Cache** | 8x256KB |
| **L3 Cache** | 20480KB |
| **Thermal design power (TDP)** | 70W |
| **Memory** | |
| **Total amount (max)** | 40GB |
| **# and size of DIMM** | 8*4GB, 4*2GB |
| **Memory characteristics** $(C70, C71)$ | 4GB (Micron): DDR3 LV, SDRAM, RDIMM (registered), ECC, single rank (1Rx4), DDR3-1600, PC3-12800, CL11<br>2GB (Qimonda): DDR3, SDRAM, RDIMM (registered), ECC, single rank (1Rx4), DDR3-1066, PC3-8500R, CL7 |
| **Disk** | |
| **Drives** | 1x73GB 15K RPM SAS |
| **Controller** | Integrated SAS controller (*Intel* C600) |
| **Network adapter** | 4xSuperFast NIC, 100Mbit |
| **Power supply unit** | |
| **Quantity and rating** | 1x450W, Delta |
| **Specification** | 94% (platinum efficiency), 100-240V, 50/60 Hz |
| **Fans** | 5, 4+1 redundant |

[321] Influence of the results: due to resource management, memory (access) management, or OS-based scheduling onto the hardware resources (load balancing, multitasking, or context switch)

We choose a Microsoft Windows OS to ensure the full support of the device states, according to the ACPI standard, considering an enabled power management. We execute Java-based[322] measurement tools and benchmarks on our server system, which may have an impact on our results because some implementations have a poor garbage collection, a large memory footprint, or a lack of garbage collection on resources. Most of the tools continuously monitor and save the sensor data[323] at the same time. In Table 48, we show the software versions of our system under test. Additionally, the system is accessible from outside by the baseboard management controller (BMC), which provides the intelligent platform management interface (IPMI) to enable the sensor tracing[324] independently of the operating system.

Table 48: System under test (SUT) – software settings

| Category | Settings |
|---|---|
| **Operating system (OS)** | |
| **Version** | Microsoft Windows Server 2012 Standard R2 |
| **Power management** | Enabled |
| **Software** | |
| **Java runtime environment (JRE)** | 1.8.0_91 |
| **Oracle VirtualBox** | 4.1.18 r78361 |
| **BIOS/UEFI (ACPI support)** | SMBIOS V2.4 |
| **Baseboard management controller** | |
| **Integrated remote management controller (iRMC)** | iRMC S4, 256 MB attached memory incl. graphics controller |
| **Firmware version** | 7.61 |
| **Sensor data record (SDR)** | 09.71 (ID 0356) |
| **Intelligent platform management interface (IPMI) version** | 2.0 |

### 7.1.2 Measurement Infrastructure

Here we consider several measurement tools to monitor the power, performance, and temperature of the target SUT, which are available for the most popular operating systems and easily accessible in the public domain. We install the software-based measurement tools on our target SUT, execute all of them in parallel, and store the results[325] afterwards.

[322] Java tools: platform-independent, must be interpreted, needs the Java runtime environment (JRE)
[323] Sensor data: hardware sensors are mounted on the motherboard
[324] Sensor tracing: read sensors (power, temperature, fan speed)
[325] Results: files in *.csv format, some tools support caching, others write data at all times

We always save all the tool information that is available during our evaluations considering a steady load of writing data onto disk. Among our repeated measurements[326], we assume a steady situation[327] in the laboratory concerning a constant ambient temperature, static air pressure, or humidity.

The *HW Monitor Pro*[328] provides the status of the component utilization levels and especially distinguishes between the processor cores and threads. The tool shows the current frequencies, which are restricted to the processor cores[329], monitors the processor package power, and uncores power[330], according to the voltage regulator module (VRM). The *HW Monitor Pro* records the temperatures of the processor cores and the core temperature of the memory modules. The tool reads the sensor data by a time base of one second and continually writes the sensor data into an appropriate file on the hard disk.

The *Intel Power Gadget*[331] is the only tool that provides the cumulative energy consumption of the input/output, processor, and memory of their aggregated power values. Herein, we particularly focus on the memory power consumption because the other tools do not provide the same level of detail. The tool does not distinguish into the particular processor cores and therefore provides the processor frequency as an all-embracing time-based vector. The *Intel Power Gadget* monitors the temperatures within the system, whereby the time base of the *Intel Power Gadget* is fixed to 100 milliseconds.

The major benefit of the *Intel Power Thermal Utility*[332] (*PTU*) is the flexible configuration of the time base, which we define as 50 milliseconds to monitor and trace the sensor data more accurately in comparison to the other tools. Therefore, we consider the processors' power, temperature, frequency, and utilization level to evaluate our processor and memory model. The tool directly accesses the processors' internal sensors and provides the voltages, which we evaluate in our processor model considering the p-states and the related voltage-frequency pair. The *Intel Power Thermal Utility* caches the sensor data, which we save in a specific log file afterwards.

[326] Repeated measurements: up to five times, evaluating either identical or modified server system configurations
[327] Steady situation: always ideal operating conditions because of the HVAC system
[328] *HW Monitor Pro*: http://www.cpuid.com/softwares/hwmonitor.html
[329] Processor cores: processing functionality and instructions, such as arithmetic logic unit (ALU), floating-point unit (FPU), L1 and L2 cache
[330] Processor uncore: integrated subsystems on the processor (on-chip interconnect or communication), e.g. the Quick Path Interconnect (QPI), power controller, integrated memory controller (IMC), I/O controller, network controller, scalable memory interface (SMI), distributed/shared last-level cache (LLC), such as L3 cache
[331] *Intel Power Gadget*: https://software.intel.com/en-us/articles/intel-power-gadget-20
[332] *Intel Power Thermal Utility*: http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/cpu-monitoring-dts-peci-paper.pdf

The *Intel* tools (*Intel Power Gadget* and *Intel Power Thermal Utility*) provide several relevant processor-specific details based upon the processor socket, family, series, and generation, which are only accessible by these tools because of the particular sensors inside the processor[333].

A special tool is called *Kalcheck*, developed by Fujitsu, which monitors the power and temperature values independently of the operating system. In particular, *Kalcheck* provides server-specific, internal system temperatures of the system-board, power supply unit, and each individual memory module. We can observe the ambient and processor-specific temperatures to analyze the thermal development within the enclosure considering the fan speed. In addition, *Kalcheck* distinguishes between the secondary (provided by the PSU) and primary (input to the PSU) power of the system, and separately monitors the system-board and hard disk power. *Kalcheck* uses IPMI over Ethernet to connect to the BMC, which accesses to sensors with the inter-integrated circuit bus (I²C). The time base is restricted to one second because the sensors are connected to the I²C bus, which has a limited bandwidth, especially in case of one hundred sensors within a particular server system. The sensor data record (SDR) inside the BMC specifies the certain addresses of the sensors, sensor types, and amount of sensors. The following table shows an overview of our measurement tools, their time basis, and extra settings.

Table 49: Measurement tools and settings

| Measurement tool | Description | Time base | Settings |
|---|---|---|---|
| ***HW Monitor Pro*** | Version 1.25.0 | $1000ms$ | Disable multi-curves |
| ***Intel Power Gadget*** | Version 3.0 | $100ms$ | |
| ***Intel Power Thermal Utility*** | Version 3.2 | $50ms$ (configurable) | Max records 99999999 |
| ***Kalcheck* (Fujitsu-specific internal tool*)*** | Date of manufacture 2014-04-01 | $1000ms$ | Poll -temp -fan -power -logfile ‘logfile.csv’ |

We execute the measurement tool *Kalcheck* on an extra computer to avoid using the loopback network interface on the target system, which may lead to additional load, and the data tracing is accessible independently of the target OS. We always start all measurement tools of the same sequence before we enable the benchmark procedures, using the remote desktop protocol (RDP). In the next step, we run the benchmarks from scratch[334] to avoid caching effects and improve the repeatability when testing. Figure 103 shows our evaluation environment in the laboratory.

[333] Processor sensors: digital thermal sensors (DTS)

[334] Start from scratch: after rebooting the SUT, access benchmark after starting the measurement tools

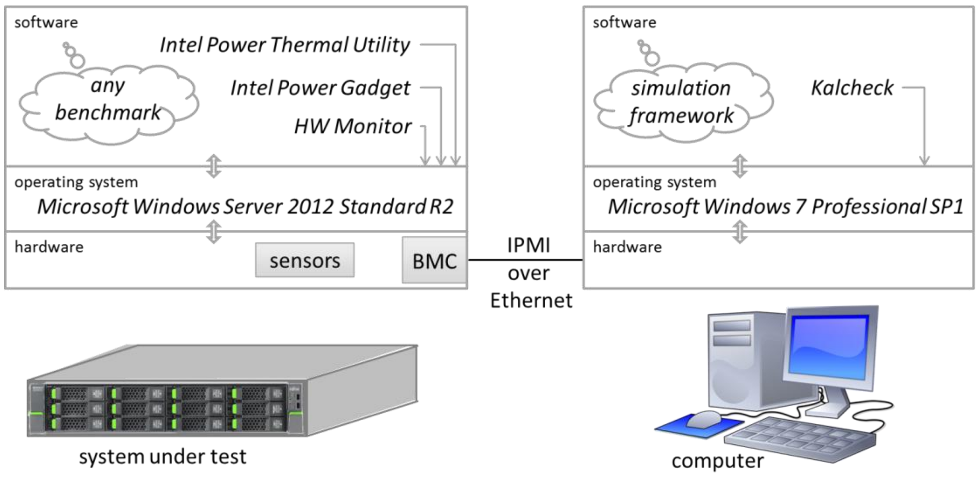


Figure 103: Evaluation and simulation environment

We share one computer to execute *Kalcheck* and start our simulation framework, including our *MFSMOS* approach as a prototype implementation. We trace the system under test using the *HW Monitor Pro, Intel Power Gadget, Intel Power Thermal Utility,* and *Kalcheck* to collect the data for our simulation framework. We disable all communications between both systems and stop every measurement tool when we start our simulation model in MATLAB. Table 50 lists the hardware and software settings of our computer.

Table 50: Computer – hardware and software settings

| Category | Settings |
|---|---|
| **Hardware** | |
| **Platform segment** | Laptop, *Fujitsu Celsius H710* |
| **Processor** | *Intel Core i7-2760QM, 2.4GHz* |
| **Memory** | 8GB, DDR3-SDRAM |
| **Software** | |
| **Operating system** | Microsoft Windows 7 Professional SP1 version 6.1 (Build 7601: SP1) |
| **Java runtime environment (JRE)** | 1.7.0_71 |
| **MATLAB** | R2015a (8.5.0.197613), 64-bit |
| **Simulink** | R2015a (8.5.0.197613), 64-bit |

The possible resolution and precision of the temperature, power, and fan speed depend on the sensor quality and sensor position within the server system. Our target SUT has approximately 200 sensors in which the sensor data record defines how to interpret the raw values. The temperature sensors are determined to an accuracy of half a degree Celsius. The power sensors of the processor, memory, and hard disk drive are more accurate ($< \pm 0.5\,W$) in comparison to the power supply sensors ($\geq \pm 4\,W$). The fan speed sensors have an accuracy of approximately $\pm 10\,RPM$. Furthermore, the time used for collecting data varies because of the flexible configuration of the sensors and their corresponding amount of data, which we save in the log files. The measurement tools use the clock of the operating systems instead of a real-time clock, which implies that the handling or triggering[335] of sensor data does not fit the configured time exactly.

### 7.1.3 Benchmarks

We consider the standard publicly available benchmarks and their metrics, such as *SPEC*[336], *TPC*[337], and *PassMark*[338]. Table 51 lists the various benchmarks and settings, which we install and execute on the target SUT to analyze multiple activity rates. We assume that the OS insignificantly limits the benchmarks, because we assign the highest priority to our benchmarks, and additionally set the affinity at all processor cores and threads to ensure that all of them are used similarly. As a result, we enhance the repeatability and restrict the possible variations between various benchmark tests.

**Table 51: Benchmarks settings**

| Benchmark | Settings |
|---|---|
| ***SPECpower*** | Version ssj2008-1.12 |
| **Operating system (OS)** | SPEC Open Doors 2006 F500 |
| **boot (Management) firmware version** | 1.2.3.4 (64-bit) |
| **Java Virtual Machine (JVM) version** | SPEC Java VM 5.0 (build 1.2.3.4-tricore 20071111) |
| **JVM command-line options** | -Xms3500m -Xmx3500m –XrunFast -XconsumeLessPower -Xmn3100m |
| **Workload version** | SSJ 1.2.10 |
| **PTDaemon** | v1.4.2 |
| ***PassMark*** | Version 8 |
| ***PassMark CPU*** | CPU mark, integer math, floating-point math, prime numbers, extended instructions (SSE), compression, encryption, physics, sorting, single threaded |
| ***PassMark Memory*** | Memory mark, database operations, read cached, read uncached, write, available RAM, latency, threaded |
| ***MemTest86*** | Version 7.0 free edition, pattern testing |

[335] Trigger the data: The measurements have non-equidistant times, e.g., $1.02\ s$ or $0.98s$ between the measurement time stamps.

[336] *SPEC*: http://www.spec.org/

[337] *TPC*: http://www.tpc.org/default.asp

[338] *PassMark*: http://www.passmark.com/

The *SPECpower*[339] benchmark measures the power and related performance while utilizing the server system at various levels, considering the processors, memories, or caches. *SPECpower* starts a calibration phase (minimum of three runs) to identify the maximum benchmark throughput. Afterwards, the sequence of the target throughput decreases from the maximal target throughput from $100\%$ down to $0\%$ in about $10\%$ of the discrete steps of the calibrated throughput. *SPECpower* tries to reach an approximate target throughput of $90\%$, for instance. The tool uses the server-side Java (*ssj*) workloads[340] that execute multiple Java virtual machines (JVMs), considering a large number of users and various transaction types. A result of the *SPECpower* benchmark is the actual target throughput, such as $90.7\%$ of the maximal throughput, which should be as close as possible[341] to the target throughput of $90\%$. We assume that the system approximately operates at a certain utilization level in average when working at a particular target throughput, which will not be a steady value[342]. The utilization level of the processor behaves proportional to the load (target throughput) of the benchmark. Furthermore, the benchmark provides the average power consumption and the number of *ssj*-operations[343] $(ssj\_ops)$ on the basis of the various target throughputs. The tool considers both aspects to calculate the performance-to-power ratio at each target throughput. We cannot use the certified (officially released) power analyzer[344] in the *SPECpower* benchmark, because of the extraordinary equipment costs to consider high-resolution power measurements. Instead, we consider various measurement tools, monitoring the power of the motherboard, processor, memory, or power supply unit. Therefore, our *SPECpower* log file does not include the average power and the performance-to-power ratio in a precise manner.

We execute the *SPECpower* benchmark in four different server configurations, which either changes the number of processors or the total amount of the memory capacity. The server system requires the balance of the memory modules among the processors, which depends upon the enabled processors and the memory channel configuration. Therefore, we consider a dual channel $(SPx.2.z)$ and a triple channel $(SPx.3.z)$ configuration, whereby $x$ denotes the number of the processors, $y$ indicates the channel configuration, and $z$ refers to the total memory capacity within the system, as shown in Table 52.

**Table 52: *SPECpower* – test cases for $(SPx.y.z)$**

| | | **# of memory modules * capacity (GB) per module** | |
|---|---|---|---|
| | | **2*4** | **3*4, 3*2** |
| **# of processors** | **1** | $SP1.2.8$ | $SP1.3.18$ |
| | **2** | $SP2.2.16$ | $SP2.3.36$ |

[339] *SPECpower*: http://www.spec.org/benchmarks.html#power
[340] SSJ workloads: Java program, https://www.spec.org/power/docs/SPECpower_ssj2008-Design_ssj.pdf
[341] As close as possible: less than $2\%$ positive or negative deviation
[342] Steady value: average value over an interval
[343] *ssj*-operations: performance indicator
[344] Power analyzer: https://www.spec.org/power/docs/SPECpower-Device_List.html

In contrast, the *TPC* suite focuses on data-centric benchmarks, which are relevant in industry and uncommon in academic evaluations. We neglect the wide range of micro-benchmarks in our evaluation because of the amount of data will increase exponentially. In principle, we may add the *TPC* benchmark results when a server system is suitable[345].

A widely used software in personal and in business applications is *PassMark*, which tests x86-based computers and their components. We consider especially the *PassMark CPU*[346] and *PassMark Memory*[347], which help comparing the relative performance of the various components. Both benchmarks consist of multiple micro-benchmarks, as shown in Table 51, which we entirely execute because this operating mode is set as a default. The *PassMark* benchmarks store the maximal scores of every micro-benchmark, provide the *mixture* (overall) score of the benchmark itself, and additionally show scores of similar configurations, such as the *Intel Xeon E5-2640v2*, *Intel Xeon E5-2650L*, and *Intel Xeon E5-2648L*. We consider the *mixture* performance result, a mix of the certain micro-benchmarks, as the default maximal performance values of the components in our simulation model. To respond flexibly to customer demands, we add the results of the micro-benchmarks to our database. Table 53 and Table 54 list the *PassMark* test cases that we analyze in our evaluation, considering the processor performance $(PCx.y.z)$ and memory performance $(PMx.y.z)$ in which $y$ denotes the amount of memory modules per processor.

Table 53: ***PassMark CPU*** **– test cases for** $(PCx.y.z)$

| | | # of memory modules * capacity (GB) per module | | |
|---|---|---|---|---|
| | | **1*2** | **1*4** | **2*2** |
| **# of processors** | **1** | $PC1.1.2$ | $PC1.1.4$ | $PC1.2.4$ |
| | **2** | $PC2.1.4$ | $PC2.1.8$ | $PC2.2.8$ |

Table 54: ***PassMark Memory*** **– test cases for** $(PMx.y.z)$

| | | # of memory modules * capacity (GB) per module | | |
|---|---|---|---|---|
| | | **1*2** | **1*4** | **2*2** |
| **# of processors** | **1** | $PM1.1.2$ | $PM1.1.4$ | $PM1.2.4$ |
| | **2** | $PM2.1.4$ | $PM2.1.8$ | $PM2.2.8$ |

[345] Server system in TPC: PRIMERGY TX300 (similar to RX200/TX200) http://c970058.r58.cf2.rackcdn.com/fdr/tpcc/fujitsu-siemens.TX300.030811.01.fdr.pdf
[346] *PassMark* CPU: http://www.cpubenchmark.net/
[347] *PassMark* Memory: http://www.memorybenchmark.net/

Moreover, we consider the *PassMark MemTest86*[348] because the internal test algorithm utilizes the memory cells more individually, as done in the *PassMark Memory* benchmark. The *MemTest86* algorithm[349] reads and writes the memory cells using a pre-defined pattern, which can start at the lowest or highest address. The algorithm includes the writing of the original data and its complement. As a last step, the tool checks the write process by reviewing the data and counting the faults. The *MemTest86* procedure increases the memory address and repeats the previous steps toward the end of the memory address range. We further analyze the memory modules $(MTx.z)$, considering the amount of processor threads $x$ and the total memory capacity $z$, as listed in Table 55. We create and reconfigure exactly the same virtual machine settings to enable the analysis of all test cases and we adapt the memory capacity by adjusting the virtual machine properties. This special virtual environment is necessary because we cannot execute the measurement tools in the stand-alone[350] memory testing software *MemTest86*.

Table 55: *MemTest86* – test cases for $(MTx.z)$

| | | Total memory capacity (GB) | | | | |
|---|---|---|---|---|---|---|
| | | **32** | **24** | **16** | **8** | **1** |
| **# of processor threads** | **1** | $MT1.32$ | $MT1.24$ | $MT1.16$ | $MT1.8$ | $MT1.1$ |
| | **8** | $MT8.32$ | $MT8.24$ | $MT8.16$ | $MT8.8$ | $MT8.1$ |
| | **16** | $MT16.32$ | $MT16.24$ | $MT16.16$ | $MT16.8$ | $MT16.1$ |

Finally, we analyze the boot phase of the server system, where $x$ is the number of the processors, $y$ is related to the channel configuration, and $z$ indicates the total memory capacity within the system, as shown in Table 56.

Table 56: Boot phase – test cases for $(BPx.y.z)$

| | | # of memory modules * capacity (GB) per module | | |
|---|---|---|---|---|
| | | **1*2** | **1*4** | **2*2** |
| **# of processors** | **1** | $BP1.1.2$ | $BP1.1.4$ | $BP1.2.4$ |
| | **2** | $BP2.1.4$ | $BP2.1.8$ | $BP2.2.8$ |

[348] *MemTest86* : http://www.memtest86.com/download.htm
[349] Algorithm: http://www.memtest86.com/technical.htm
[350] Stand-alone software: *MemTest86* boots from a USB flash drive or CD

In the initial starting phase, the BIOS/UEFI checks whether the components work properly. Therefore, each component will be accessed during the initialization and boot process at some specific moment, which usually results in the maximum possible power consumption within some seconds or up to a maximum of a few minutes. The power consumption behavior can be compared with the spin-up process of a hard disk in which we require more power to spin the disk from $0RPM$ to $7200RPM$ than to keep them rotating [HSR et al. 2008].

Table 57 lists the execution time statistics of the various benchmark runs considering the repetitions because all benchmarks vary at each run. The *SPECpower* benchmark requires approximately 77 minutes, followed by the *MemTest86* at approximately 27 minutes. In contrast, the *PassMark* benchmarks perform within about three to four minutes, which is nearly as fast as the boot phase of the server system of less than three minutes. We cannot guarantee the exact benchmark behavior, but evaluate an explicit benchmark trace.

Table 57: Execution time statistics – benchmarks

| | | Benchmarks | | | | |
|---|---|---|---|---|---|---|
| | | *SPECpower* $SPx.y.z$ | *PassMark CPU* $PCx.y.z$ | *PassMark Memory* $PMx.y.z$ | *MemTest86* $MTx.z$ | *Boot phase* $BPx.y.z$ |
| **Execution time [min]** | **Maximum** | *84.47* | *4.09* | *3.21* | *28.18* | *3.18* |
| | **Minimum** | *73.56* | *4.02* | *2.92* | *26.65* | *2.23* |
| | **Mean** | ***76.32*** | ***4.04*** | ***2.98*** | ***27.24*** | ***2.70*** |
| | **Median** | *74.24* | *4.03* | *2.95* | *26.87* | *2.73* |
| | **Standard deviation** | *4.63* | *0.02* | *0.10* | *0.83* | *0.27* |
| | **Variance** | *21.46* | *0.0005* | *0.01* | *0.69* | *0.07* |

### 7.1.4 Measurement Issues and Restrictions

Before we present the evaluation results, we give a general overview of our general findings when analyzing the measurements of the real system. First, the results of the benchmarks and measurement infrastructure have to be consistent over time because the tools do not use a real-time clock or have the same time stamps. Thus, we convert all time stamps of the various signals into a second-based format and slightly shift some signals (in milliseconds) to synchronize them, considering the different date and time formats. In addition, we configure a particular start time and stop time as a common base to provide valid time stamps in our analysis at any time. We specify $'0s'$ as an absolute and common start time of all signals in every experiment, which builds the basis of the related times on the x-axis represented with an exponent.

Figure 104 exemplarily shows the measurements of the *Intel Power Thermal Utility* recording the processor frequency while executing the *PassMark CPU* benchmark ($PC1.1.2$). In our frequency measurement, we observe multiple frequency peaks that are higher than the maximal possible frequency of the processor defined at $2300MHz$, such as $3033MHz$ at a time[351] stamp $t = 5.052 * 10^6 s$. We assume that all signals during our measurements are affected by such measurement errors. We especially analyze the appearance of the incorrect frequencies in relation to the total amount of occurring frequencies and observe nearly $+0.33\%$ at the *PassMark CPU* and approximately $+0.25\%$ at the *SPECpower* benchmark in the mean. The upward extreme outliers[352] are not the only measurement errors; we also found values that sharply vary from the rest in a group of observations under the same conditions. We observe a couple of measurement errors that just occur for a short period in relation to the entire benchmark (unusually small). Moreover, Figure 104 presents a certain measurement error between the time period $T = [5.048 * 10^6, 5.052 * 10^6]s$, which shows a non-signal. Such a reading error may occur in further measurements. Even more, we face a problem with zero values in our evaluation that leads to infinite values when we calculate the accuracy because of a division by zero.

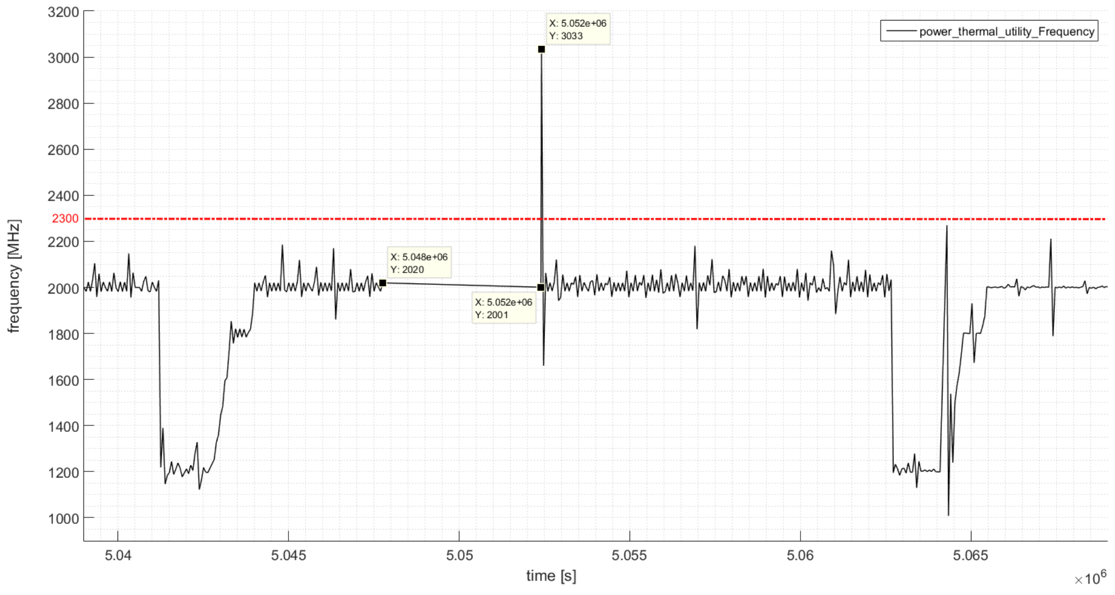


**Figure 104: *PassMark CPU* $(PC1.1.2)$ – measurement accuracy**

[351] Time notation: $1 * 10^5 s$ corresponds to `1e+06s` MATLAB notation. We simplify the time notation for better readability and define $1 * 10^6 \equiv 1e + 06$, which ensures easy reading in our graphs and analysis.

[352] Detect outlier: exceed the minimum/maximum limit (e.g. frequency) or calculate the mean $\mu$ and variance $\sigma$. Outliers are either less than $\mu - 3\sigma$ or greater than $\mu + 3\sigma$, known as three sigma rule in normal distribution; our MATLAB version does not support the current 'isoutliers' function.

We require a relatively long period in comparison to the measurement errors in order to achieve a sufficiently high density of samples that ensure the statistical significance of our signals. The *SPECpower* benchmark finishes after around 77 minutes, the *PassMark CPU* requires approximately four minutes, the *PassMark Memory* takes nearly three minutes, the *MemTest86* stops after 27 minutes, and the boot phase ends after three minutes, as shown in Table 57.

As stated in Section 7.1.2, we execute our simulation framework on a common computer and list the related settings in Table 50. The measurement infrastructure (hardware and software) produces a systematic error because of their uncertainty based on the limited accuracy (the digits in each measurement) or imprecise calibration [Rab 2010]. Another systematic error is the environmental condition, which may change during our measurements, such as an increasing ambient temperature. We are restricted to the specific ambient temperature range of $20 - 25°C$ because of the static HVAC settings in our laboratory. We cannot adjust the measurement environment of the system under test, as the changes are possible in our simulation runs. Additionally, we assume that a highly utilized processor, which is located relatively nearby the memory module on the motherboard, may be another disturbance of the thermal development within the system.

## 7.2 Analysis of the Aspect-based Calculation Methods Regarding Their Accuracy

### 7.2.1 Objective

Our aim is a general verification as well as evaluation of the concepts' operating principles and our related calculation methods, which have to react on heterogeneous workload scenarios. We analyze the accuracy of our aspect-based component models and check whether our approach is adequate. The abstraction level of the server system and its components should be low enough to support architectural and structural changes at the physical domain. We want to find the limits of our model because of the chosen abstraction level to treat the conditions of the vendor and the customer-specific demands at the same time. Additionally, our component-based models shall improve the power calculation of the commercial tools and avoid over-provisioning.

### 7.2.2 Evaluation Criteria

Our models immediately need to react upon the synthetic workload scenarios (category-specific utilization levels) provided by the commercial tools. We assume that if our models handle the flexible category-specific utilization levels step-by-step, the entire simulation model will respond adequately at all possible utilization levels. In this analysis, we answer the following question:

- How much can we trust our aspect-based calculation methods and component-based models in our simulation?

As a prerequisite, we have to answer the question of how to determine whether the aspect-based calculation methods are accurate. We study and trace the particular components within the system to verify our concept and show the plausibility when reaching extreme values. The various commercial tools primarily consider spreadsheets, instead of measuring the real server systems, which saves costs. We evaluate the accuracy of our aspect-based calculation methods by comparing our results with the measurement values gained from the real server system. Therefore, our evaluation criteria are the *absolute* and *relative differences* between the simulation-based values and the measurement results at the various utilization levels. When simulating the power consumption, we require an over-prediction of less than $+10\%$ in relation to the real-life measurement. We concentrate on the power and thermal measurement in our analysis and avoid any vendor-specific hardware to show the applicability of several generic components.

### 7.2.3 Experimental Setup

We specify the general evaluation environment and measurement infrastructure in the Sections 7.1.1, 7.1.2, and 7.1.3. In Section 7.2, we describe the experimental setup of the *accuracy analysis* in which we exclusively study an audited hardware, because we can crosscheck our simulation-based results with the empirical measurements gained from the real hardware, considering identical data of the components. We analyze two *Intel* processors of the same family and a couple of memory modules that have various capacities and originate from a certain manufacturer. We separately evaluate the category-specific components concerning their accuracy of the aspect-based calculation methods, which is possible because of our modular and hierarchical concept. Figure 105 shows the block diagram of the controller layer in our simulation model in which we explicitly simulate the memory module, for instance.

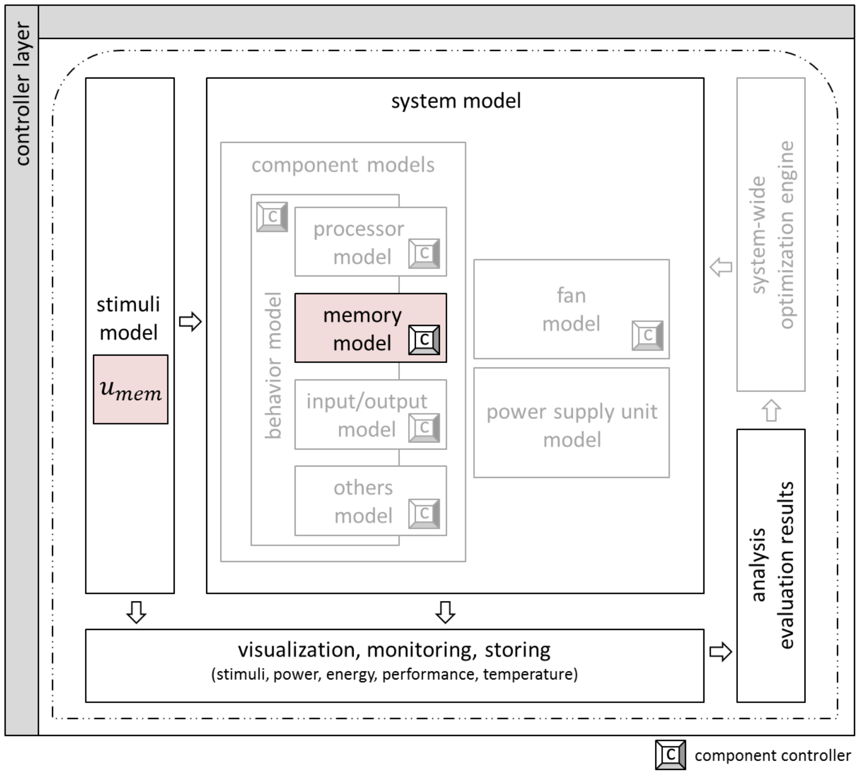


Figure 105: Controller layer – block diagram considering an exclusive memory model for evaluation purposes

We provide the memory utilization level $u_{mem}$ to our memory model within the system model and calculate the various aspects of the memory module, which we further analyze in our controller during the simulation. Herein, we exclusively consider the memory component and call the particular *calc_category()* method, provided by the memory model, which is isolated and encapsulated from the other components, see Section 6.2. We neglect the complex simulation model (e.g. the thermal control) and especially analyze the component behavior itself and its internal aspect-based relations, as shown in Figure 44, concerning each utilization level. In this evaluation section, we disable the *system-wide optimization engine* because we analyze the results of our component-based models.

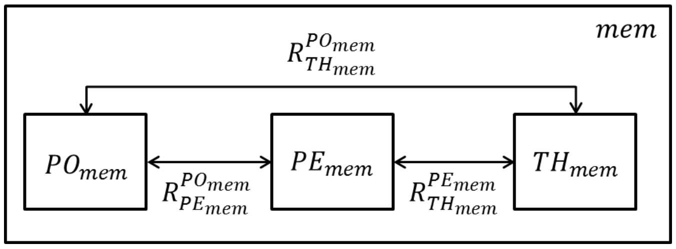

$mem$
$R_{TH_{mem}}^{PO_{mem}}$
$PO_{mem}$
$PE_{mem}$
$TH_{mem}$
$R_{PE_{mem}}^{PO_{mem}}$
$R_{TH_{mem}}^{PE_{mem}}$

Figure 106: Aspect-based memory module and corresponding relations $(R_A)$

Therefore, we restrict the stimuli of our simulation model with regard to the memory and set the remaining utilization levels to zero[353], which results in ignoring the other components in the simulation, as shown in Figure 107. Moreover, we ignore the relations or communications between the various components to isolate and encapsulate the memory module from the other components. We separately study the memory component, as if the memory model was detached[354] from the entire simulation framework.

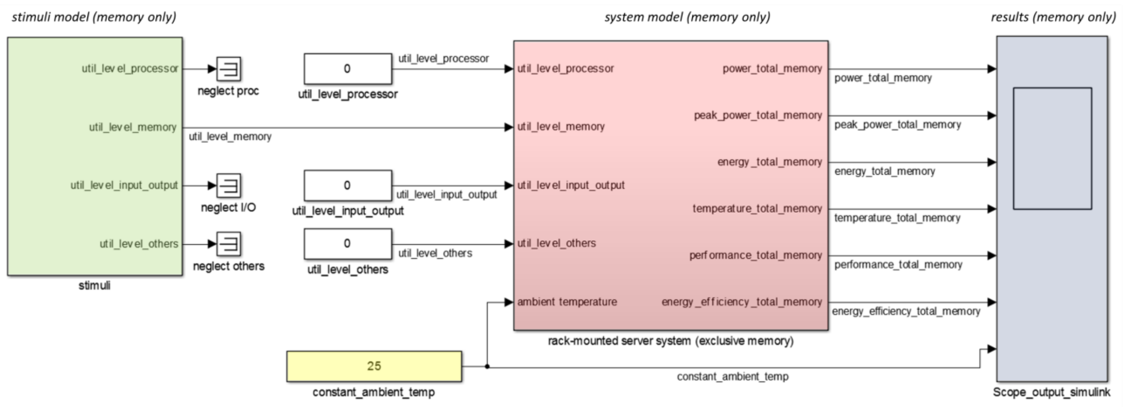


Figure 107: Simulation model – stimuli and server system for the exclusive memory evaluation

We choose this approach to simplify the evaluation and speed up the simulation in which we can easily import the utilization levels of the real-life measurements as an input parameter of our component-specific aspect-based calculation methods. We can effortlessly include the measurement trace[355] that is the subject of our analysis and calculate the *absolute* as well as a *relative difference* regarding our simulation-based results. We consider the following identical data as the basis of our simulation and internal aspect-based calculation methods as input parameters[356]:

- Characterization of the component (e.g. category, static configuration, technical specification, and dynamic characteristics)
- Ambient temperature
- Utilization level

First, we study a certain memory module and a particular processor, an audited hardware configuration, considering a steady technical specification under varying utilization levels. We execute and analyze various benchmarks to cover the entire range of the utilization levels of

[353] Set utilization levels to zero: synthetic constraint, which is unrealistic in a total server system because the most used components are always very small

[354] Detaching single components: Each category has its certain interface, including the aspect-based calculation methods, thus each component model is available in a standalone model, if necessary.

[355] Include the measurement trace: no need of additional synchronization effort, import time-based MATLAB vectors

[356] Input parameter: same parameters required within the simulation framework and *calc_category()* method

especially the single component in our memory evaluation because we cannot restrict the usage towards a particular level[357], as is possible in the processor evaluation. We essentially analyze the *PassMark Memory* and *PassMark CPU* regarding their memory-specific utilization levels to reflect the certain utilization levels. Additionally, we consider further details of the workload, such as the low-level observations when reading/writing the data into the memory cells or executing different instructions. We start with the power simulation and afterwards analyze the thermal development.

We consider the *SPECpower* benchmark to explicitly adjust the utilization levels of the processor in equidistant steps of $10\%$ in the interval $[0,100]\%$, which is nearly proportional to the calibrated throughput. We execute the standardized benchmarks to ensure the reproducibility because of their synthetic workload scenarios, see Section 7.1.3. In the beginning of our evaluation, we do not change the technical specification, as described in this section. In the next section, we analyze the memory modules and processors, considering the technical specification tree of each component, and analyze the effects upon the changing component characteristics.

### 7.2.4 Results and Analysis

We exemplarily present the results of our aspect-based component models to evaluate our concept and especially consider the memory and processor, concerning their power consumption and temperature development. We analyze the aspect-based accuracy, the *absolute difference* between the measured and simulated values (in the mean) and state their *relative differences* (in the mean) with regard to the flexible utilization levels that influence the component states.

***Memory Power Evaluation***

A key performance metric of the memory module is the actual utilization level that indicates the physical usage of a memory module, usually expressed in percentages. The utilization level has the major impact on the memory power consumption and corresponding thermal development. We conjecture that simulating the memory power on the basis of the utilization level is probably imprecise. If the memory power is inadequate, we will investigate and analyze the effects of the memory read-to-write ratio.

First, we exemplarily analyze a $4GB$ memory module of *Micron* ($LV\ DDR3-SDRAM$, see Table 47), which we assemble together with one enabled *Intel Xeon E5-2650v2* processor in our system under test. Our chosen memory module is nearly identical with the memory module in our database that we consider as a basis (metadata default) to simulate the power consumption. We avoid the characteristic changes in this section, which we further analyze in Section 7.3, but consider the technical specification in our aspect-based calculation method at the same time, such as the *vendor*, *die*, or *family*.

[357] Particular levels: We cannot generate utilization levels in equidistant steps.

We execute the *PassMark Memory* benchmark ($PM1.1.4$) on the real server system and trace the memory utilization level (by the *HW Monitor Pro*), power consumption (by the *Intel Power Thermal Utility*), and temperature (by the *Intel Power Gadget*). The upper graph in Figure 108 shows the utilization levels while executing the *PassMark Memory* benchmark, more particularly, the data operations, reading/writing the data, or refreshing the memory cells. In our example, the *PassMark Memory* benchmark utilizes the memory module between $22\%$ and $29\%$, which we similarly observe in the remaining ($PMx.y.z$) benchmark runs. We observe that the memory power consumption reaches only predefined values during the benchmark, such as $[1.1 - 1.2]W$, $[2.5 - 2.6]W$, $[3.7 - 3.8]W$, or $[4.9 - 5.1]W$, as shown in the lower graph. The lower power values ($[1.1 - 1.2]W$) refer to the background power while precharging the memory cells. On the other hand, the memory cell selection (bank or row address) for storing the data consumes more power ($[2.5 - 2.6]W$) than the precharging process. The explicit memory operations (read or write access) result in the highest power[358] values of $[3.7 - 5.1]W$ that depends upon the concrete instruction. In contrast, the benchmark in ($PM1.1.2$) utilizes the memory module in a range between $39\%$ and $52\%$, whereby we observe similar steady power states and comparable power values in relation to the memory capacity.

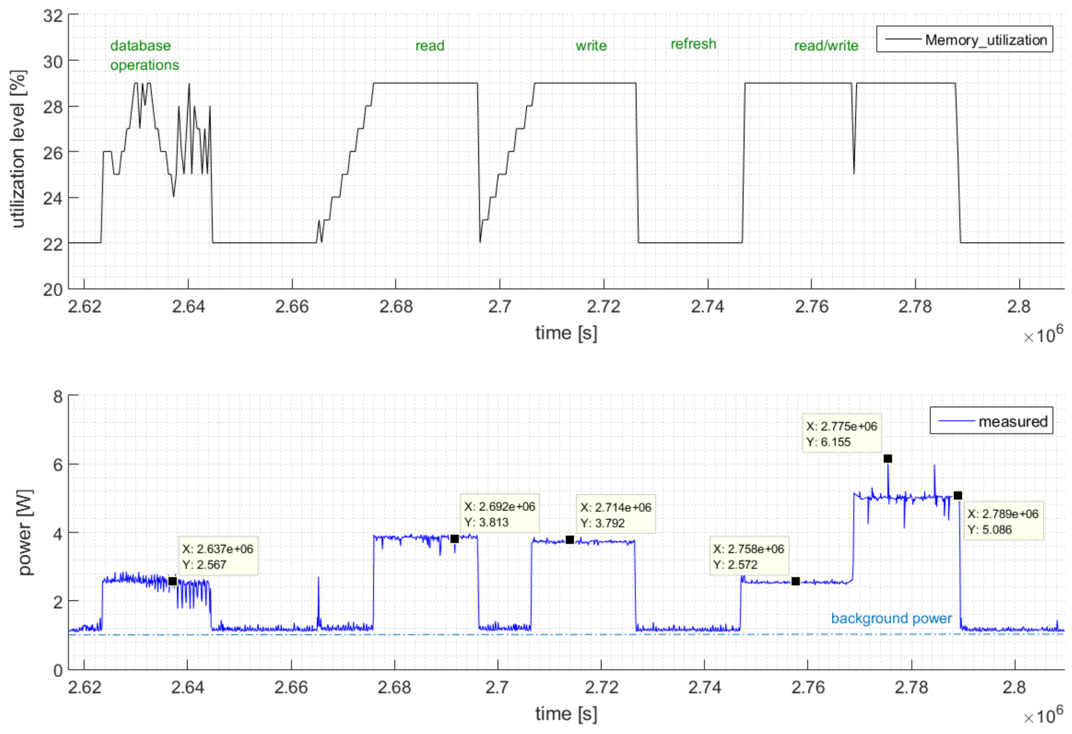


Figure 108: Memory power measurement ($PM1.1.4$)

[358] Memory power: additional information is provided in Section 3.4.2 and Section 5.2.2

In our memory power simulation, we consider the same trace of the utilization levels that we gained from our experimental measurements in $(PM1.1.4)$. Figure 109 shows our simulation-based results of the memory power (dashed-dotted red line) in comparison to the measurement trace of the *Intel Power Thermal Utility* (solid blue line) in the middle of the graph. The lower graph of the figure shows the *absolute difference* in the mean[359] (marked in purple) between the measurements and our simulation in the range of $[-3.9, +2.2]W$, which looks like an unacceptable result of our model. The extreme inaccuracy occurs in a situation when the utilization level increases from $25\%$ to $26\%$ at $t = 2.671 * 10^6 s$ $(t = 2.702 * 10^6 s)$ or decreases from $26\%$ to $25\%$ at $t = 2.788 * 10^6 s$, whereby both levels correspond to different memory states[360]. We can avoid these situations when we detect the increasing utilization levels and postpone the power consumptions, or adjust the limits of the specific states. Another reason of the inaccuracy at $T = [2.624 * 10^6, 2.644 * 10^6]s$ or $T = [2.768 * 10^6, 2.788 * 10^6]s$ is that we only specify three memory states in our non-linear calculation method to reduce the modeling effort.

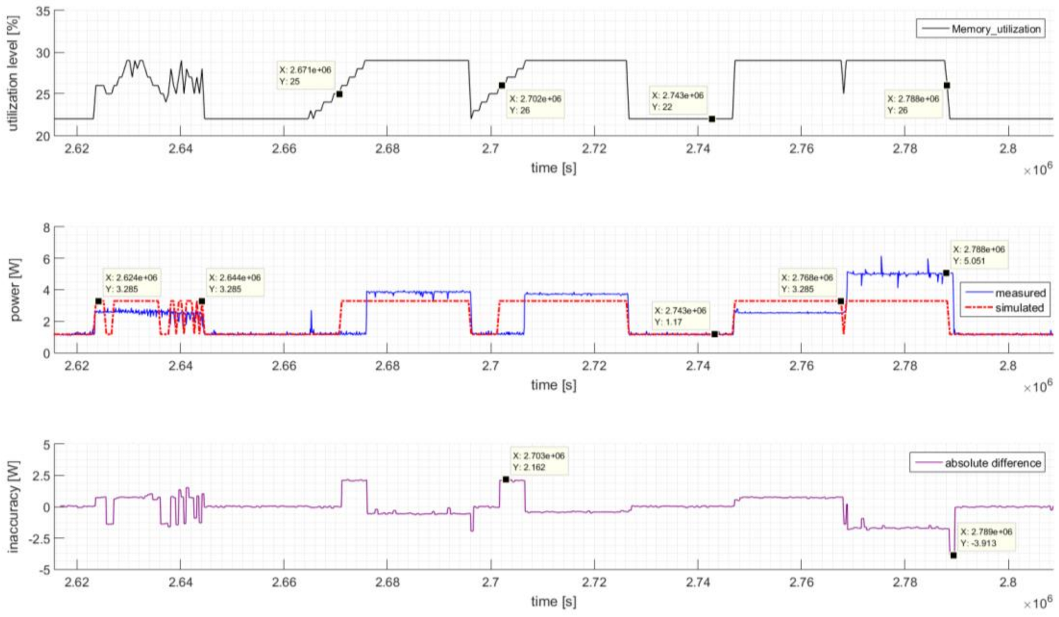


Figure 109: Memory power accuracy $(PM1.1.4)$

[359] Mean values: because of sampling, shown in Figure 111

[360] Memory states: cluster into three IDD states

On the other hand, we underestimate the power consumption by a mean inaccuracy of approximately $-0.09W$, which is an $+6.7\%$ inaccuracy in relation to our measurements during the entire benchmark in this example. The time of our absolute over-estimation is extremely low in comparison to the total time of the entire benchmark and therefore we assume that the inaccuracy is negligible. Figure 110 presents the memory power consumption by the related normalized probability function (histogram) on the basis of the *absolute differences* in Watt. In this example, we underestimate nearly half of our memory power consumption but observe a median at $\pm 0W$, a standard deviation by $+0.91W$, and a variance of $+0.83W$.

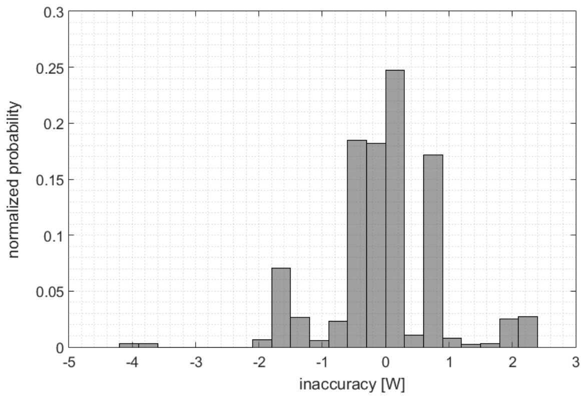


**Figure 110: Memory power accuracy – a histogram (normalized probability) $(\boldsymbol{PM1.1.4})$**

Figure 111 illustrates the identical *absolute difference* signal, which is the lowest graph in Figure 109 (marked in purple) but in this figure exemplarily limited by the horizontal axis in $T = [2.6432 * 10^6, 2.6452 * 10^6]s$, which is a simplified representation of the signal. In our accuracy analysis, we calculate the *absolute difference* on the basis of the samples of the utilization levels in equidistant steps (time stamps of the green bars) and discrete points of time of the measurements. We simplify the representation of the *absolute difference* signal into a constant signal (solid purple line) that is suitable for a fast and easy representation.

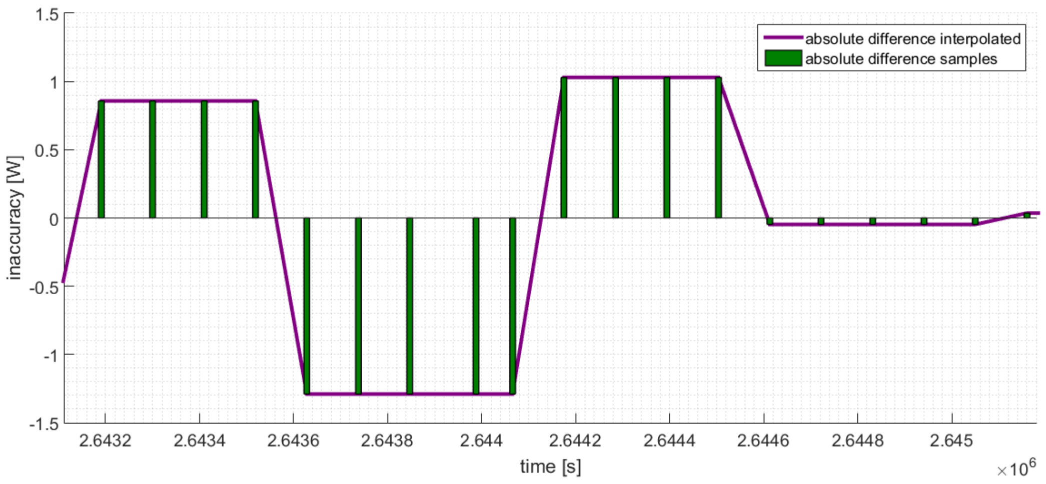


**Figure 111: Memory power – an interpolation between the samples $(\boldsymbol{PM1.1.4})$**

In the next section, we check whether we can trust the steady power states gained from the *PassMark Memory* benchmarks, which we use in our memory evaluation. For this purpose, we analyze the memory utilization level in another benchmark to detect the same memory states and show the plausibility of the measured values. We trace the memory utilization level in the *PassMark CPU* benchmarks that concentrate upon the processor utilization. The top graph in Figure 112 exemplarily shows the memory utilization level (traced by the *HW Monitor Pro*) of the identical server configuration used in $(PM1.1.4)$, but we execute the *PassMark CPU* benchmark $(PC1.1.4)$ instead. The graph in the middle of the figure illustrates the measured power consumption (by the *Intel Power Thermal Utility*) displayed by a solid blue line, and the simulation results presented by a dashed-dotted red line. We observe that the memory power consumption significantly increases approximately up to $5.2W$ when the processor searches *prime numbers*, simulates *physics interactions*, or *sort strings*[361], probably because of the read/write necessity at $T = [2.133 * 10^6, 2.152 * 10^6]s$ and $T = [2.221 * 10^6, 2.262 * 10^6]s$. At the same time, the memory utilization level does not substantially rise $(> \pm 5\%)$ in the prime numbers test, but approximately doubles in the two remaining higher utilization phases. We can observe a direct correlation between the utilization levels and the power consumption, but we consequently assume that the memory power consumption does not only rely on the utilization levels. In addition, we observe that the power consumption at the steady utilization level of $22\%$ toggles between $1.17W$ at $t = 2.154 * 10^6 s$ and $2.60W$ at $t = 2.180 * 10^6 s$, as exemplarily tagged in Figure 112. We found that the same utilization levels contradictorily result in diverse memory power values $PO_{mem}(u_{mem})$ when executing different benchmarks, such as $(PM1.1.4)$ or $(PC1.1.4)$, as shown in the following equations:

$$u_{mem} = 22\%, \{CL_{mod}, CL_{tec}, CL_{map}\} = const \tag{7.1}$$

$$PO_{mem}(u_{mem}) = \begin{cases} \sim 1.17W, \ if \ (PM1.1.4), \ \text{Figure 109} \ \ at \ t = 2.743 * 10^6 s \\ \sim 2.60W, \ if \ (PC1.1.4), \ \text{Figure 112} \ at \ t = 2.180 * 10^6 s \end{cases} \tag{7.2}$$

In fact, we cannot trust the steady state values of a particular benchmark. Accordingly, we measure and evaluate both benchmarks in our analysis considering the accuracy. We are convinced that the processor-based workload reacts on the memory modules in a similar manner when executing the database operations in the *PassMark Memory* workload, because the processor partly reads/writes data into the memory cells and loads the data considering the processor caches. In this example, the mean inaccuracy is approximately $+1W$, in which is an $+31\%$ inaccuracy in relation to our measurements if we purely consider the memory utilization levels independently of the workload scenario. Consequently, we determine the category-specific workload scenario in our simulation model when simulating the memory power consumption.

[361] Strings: single-byte characters

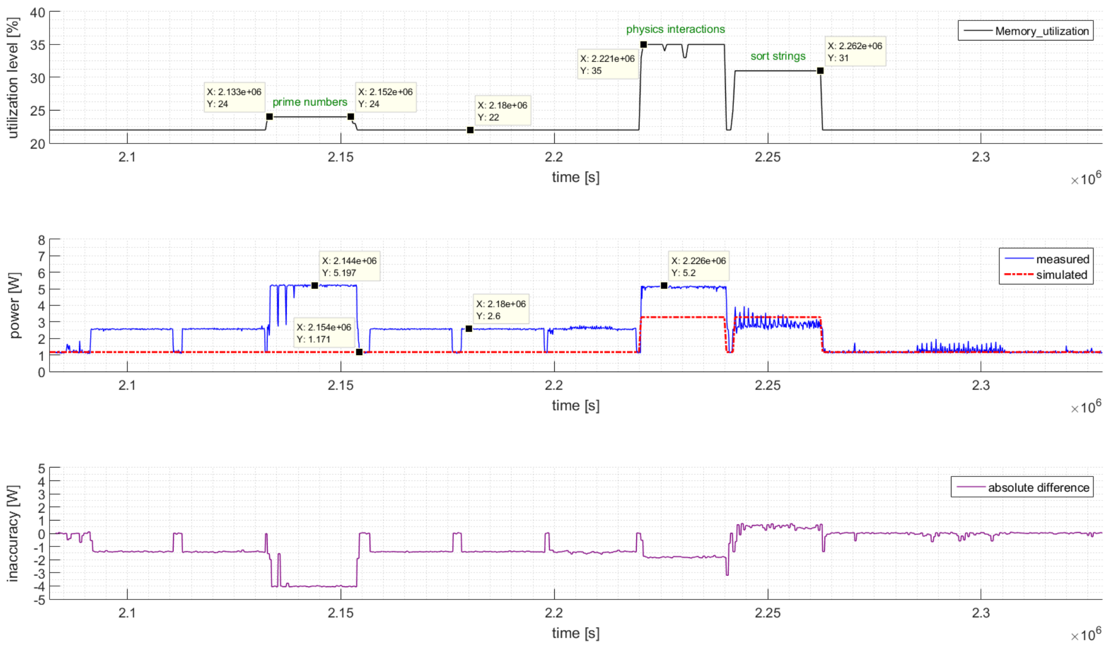


**Figure 112: Memory power accuracy $(PC1.1.4)$**

A restriction of our memory evaluation is that all of our *PassMark Memory* or *PassMark CPU* benchmarks do not fully utilize our memory modules, as exemplarily represented in Figure 109 and Figure 112. For this purpose, we briefly analyze other benchmarks[362], such as the *PMemTest, Memload, or NTMemTest*, as shown in Figure 113. In our previous examples, the memory module has a peak value of approximately $5.2W$, see Figure 112. Accordingly, to our additional evaluation we have to update the maximal power consumption of our memory module in $(PM1.1.4/PC1.1.4)$ at nearly $6.8W$. These experimental benchmark runs show the significance of the specific memory workload that we have to determine when simulating precise power values.

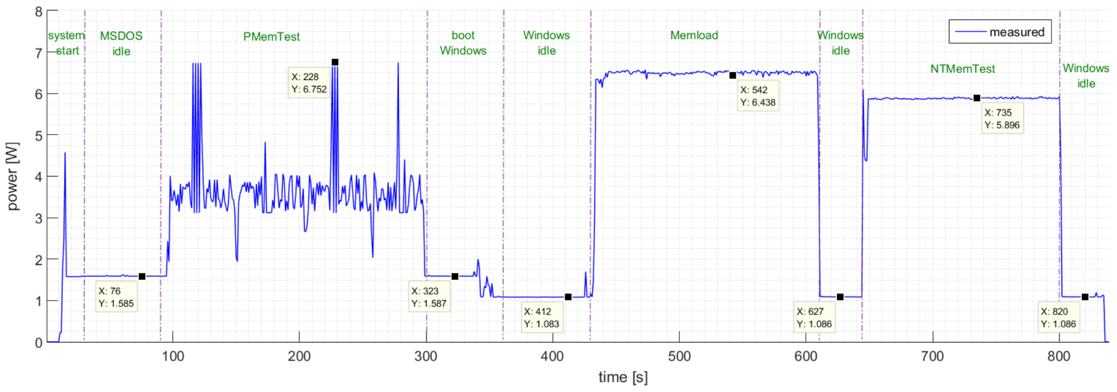


**Figure 113: Memory power consumption at various benchmarks – Fujitsu-specific hardware adapter**

[362] Other benchmarks: *PMemTest* (physical memory test)*, Memload, or NTMemTest* are specially adopted tools of the server system, but from the public sources. Fujitsu uses a special memory slot adapter that indicates the microampere, which is independent of any software.

In our memory evaluation, we neglect the explicit workload scenario and consider the memory model concerning the aspects themselves. Table 58 lists the inaccuracy of our memory power simulation under various benchmarks regarding their *absolute* as well as *relative differences* between the simulated and measured values, both stated as mean[363] values. Our aim is an error rate less than ten percent, which we specify as precise enough, see Chapter 4. We can argue that our results of the *SPECpower* benchmark are sufficient because the inaccuracy of nearly $-12.3\%$ may occur on the basis of the error propagation when simulating multiple memory modules. Our power simulation is especially inadequate concerning the *PassMark CPU* benchmarks ($PCx.y.z$), which have an approximately high inaccuracy (*relative difference*) in comparison to the *PassMark Memory* benchmarks ($PMx.y.z$), excluding the ($PM2.1.4$) run. We found that our *PassMark Memory* results are reliable and adequate because the *relative differences* are in the range of $[-10.2, +6.7]\%$ when we neglect the results of ($PM2.1.4$). In our ($PM2.1.4$) benchmark, we expect the same tendency, such as in the remaining *PassMark Memory* benchmarks, but we observe a nearly steady power value of the entire period of nearly $2.5W$. We do not trace a power increase by more than $1.5W$ that occurs in more than a few seconds, which leads to the assumption that the memory sensors produce incorrect power values of all our iterations in our evaluation. Furthermore, we monitor a random signal at the end of the benchmark that does not rely on any utilization level or instruction.

Table 58: Memory power accuracy – the simulated vs. the measured results

| | | Inaccuracy | |
|---|---|---|---|
| | | *Absolute difference* [W] (mean) | *Relative difference* [%] (mean) |
| ***SPECpower*** | $SP1.2.8$ | *-0.29* | *+4.4* |
| | $SP2.2.16$ | *-0.84* | *-12.3* |
| ***PassMark CPU*** | $PC1.1.2$ | *+0.11* | *+13.9* |
| | $PC1.1.4$ | ***-1.02*** | ***-30.9*** |
| | $PC2.1.4$ | *-0.07* | *-0.1* |
| | $PC2.1.8$ | ***-0.65*** | ***-25.1*** |
| | $PC2.2.8$ | ***-0.63*** | ***-16.0*** |
| ***PassMark Memory*** | $PM1.1.2$ | *-0.16* | *+3.6* |
| | $PM1.1.4$ | *-0.09* | *+6.7* |
| | $PM2.1.4$ | ***+1.59*** | ***+68.8*** |
| | $PM2.1.8$ | *-0.16* | *-3.3* |
| | $PM2.2.8$ | *-0.37* | *-10.2* |

[363] Mean values: $\mu$, average of all data values

Moreover, we cannot trust our memory power simulation when we neglect the workload scenario, because the simulation results are insufficiently precise, especially of the *PassMark CPU* benchmarks. The *relative difference* is in the range of $[-30.9, +13.9]\%$, which simply is based upon the utilization levels. We analyze whether the read-to-write ratio, considered as an additional impact factor, may improve the memory power simulation. Furthermore, we assume that the interactions between the processor and memory cause effects of the read-to-write ratio, especially in the *PassMark CPU* benchmarks, which results in inaccurate values because we only consider the utilization level as an input parameter of our aspect-based calculation method. As a result of our evaluation, we analyze and consider the read-to-write ratio on the basis of the interactions besides the utilization levels. Therefore, we define the read-to-write ratio assuming the benchmark specification, which refers to the certain tests, and on the other hand use the Fujitsu-internal traces[364] of the memory accesses.

Figure 114 exemplarily shows the results of our memory power simulation considering the read-to-write ratio (dashed magenta line) and the results purely on the basis of the utilization levels (dash-dotted red line) at the *PassMark CPU* benchmark $(PC1.1.4)$ in the middle of the graph. We especially adjust the read-to-write ratio when executing the prime numbers, physical interactions, or string sort algorithms, as graphically presented in Figure 112. In our example, we simulate more precisely the power values (*absolute difference*), such as by reducing the mean and the median by approximately one Watt.

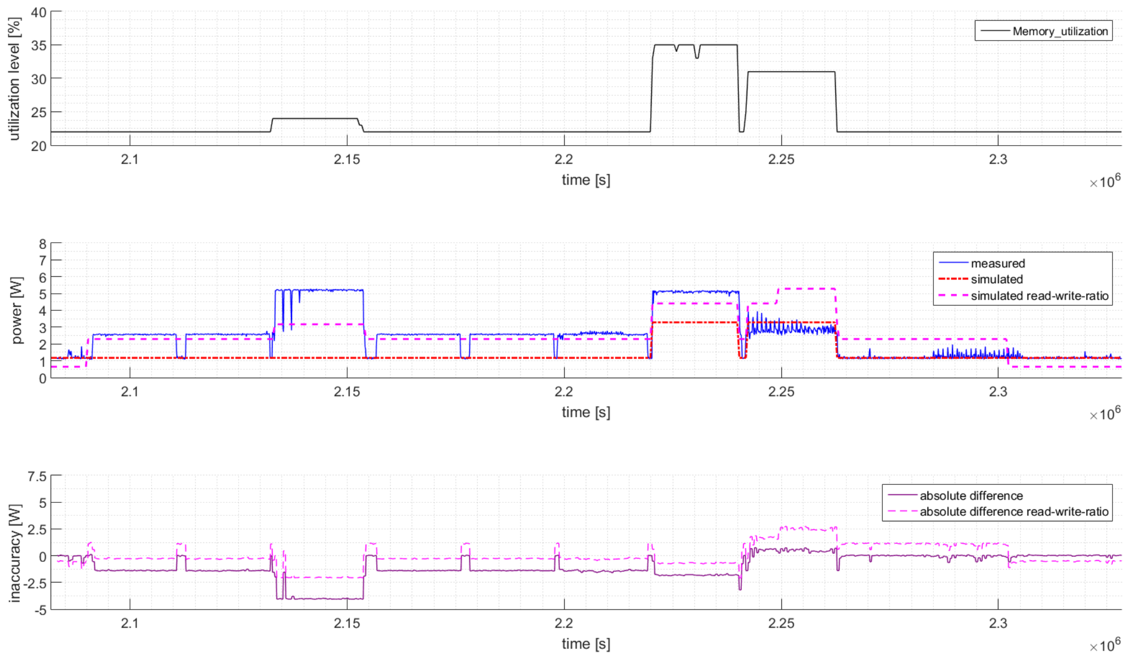


**Figure 114: Memory power accuracy $(PC1.1.4)$ considering the read-to-write ratio**

[364] Fujitsu-internal traces: specific to each memory module (vendor, capacity) and benchmark, traces are not publicly available

Our aim is to simulate the memory power as exactly as possible, such as when the ideal *absolute difference* is $\pm 0W$ and the *relative difference* is $\pm 0\%$. We determine the improvement of the read-to-write ratio by identifying the total amount of the difference to zero and subtract the inaccuracy value of the utilization level results considering the read-to-write ratio. Herein, negative improvement values refer to deterioration, see Equation (7.6).

$$|inaccuracy_{utilization\ level}\ (PC1.1.2)| = |0.11W| = 0.11W \tag{7.3}$$

$$|inaccuracy_{read-write-ratio}\ (PC1.1.2)| = |-0.15W| = 0.15W \tag{7.4}$$

$$improvement = |inaccuracy_{utilization\ level}| - |inaccuracy_{read-write-ratio}| \tag{7.5}$$

$$improvement = 0.11W - \ 0.15W = \ -0.04W \tag{7.6}$$

Table 59 lists our results and shows the reductions of the *absolute* and *relative differences* (mean values) in all simulations of the *PassMark CPU* benchmarks considering the read-to-write ratio, which we choose because we assume the highest potential for improvements. We reduce the *absolute difference* by approximately $+0.41W$ overall and the *relative difference* by around $+13.52\%$. A negative exception regarding the relative improvement builds the results in $(PC2.1.4)$ in which we increase the *absolute difference*, but with an acceptable *relative difference* by nearly $-3\%$.

Table 59: Memory power accuracy considering the read-to-write ratio

| | Inaccuracy | | Inaccuracy considering the read-to-write ratio | | Improvement | |
|---|---|---|---|---|---|---|
| | *Absolute difference* [W] (mean) | *Relative difference* [%] (mean) | *Absolute difference* [W] (mean) | *Relative difference* [%] (mean) | *Absolute difference* [W] (mean) | *Relative difference* [%] (mean) |
| $\boldsymbol{PC1.1.2}$ | ***+0.11*** | *+13.9* | ***-0.15*** | *+2.3* | ***-0.04*** | *+11.6* |
| $\boldsymbol{PC1.1.4}$ | *-1.02* | *-30.9* | *-0.01* | *+10.2* | *+1.01* | *+20.7* |
| $\boldsymbol{PC2.1.4}$ | *-0.07* | *-0.1* | *+0.16* | *+3.1* | *-0.09* | *-3* |
| $\boldsymbol{PC2.1.8}$ | *-0.65* | *-25.1* | *-0.06* | *+2.5* | *+0.59* | *+22.6* |
| $\boldsymbol{PC2.2.8}$ | *-0.63* | *-16.0* | *-0.03* | *-0.3* | *+0.6* | *+15.7* |
| **Mean** | *-0.45* | *-11.64* | *-0.02* | *+3.56* | ***+0.41*** | ***+13.52*** |

Furthermore, we improve the accuracy (median[365], standard deviation[366]) and precision[367] (variance) of our power simulation considering the read-to-write ratio during the specific benchmarks, as shown in Table 60 and Table 61. We improve the median by approximately $+0.74W$, the standard deviation by around $+0.07W$, and the variance by nearly $+0.17W$ in the mean.

Table 60: Memory power accuracy and precision statistics considering the read-to-write ratio

| | Inaccuracy | | | | Inaccuracy considering the read-to-write ratio | | | |
|---|---|---|---|---|---|---|---|---|
| | Absolute difference [W] | | | | | | | |
| | mean | median | standard deviation | variance | mean | median | standard deviation | variance |
| $PC1.1.2$ | +0.11 | +0.65 | +1.38 | +1.91 | -0.15 | +0.13 | +1.14 | +1.29 |
| $PC1.1.4$ | -1.02 | -1.37 | +1.17 | +1.37 | -0.01 | -0.29 | +1.05 | +1.11 |
| $PC2.1.4$ | -0.07 | -0.25 | +1.04 | +1.09 | +0.16 | +0.09 | +1.14 | +1.29 |
| $PC2.1.8$ | -0.65 | -1.31 | +0.79 | +0.63 | -0.06 | -0.28 | +0.72 | +0.52 |
| $PC2.2.8$ | -0.63 | -1.63 | +1.75 | +3.05 | -0.03 | -0.71 | +1.73 | +3.01 |
| **Mean** | -0.45 | -0.782 | +1.23 | +1.61 | -0.02 | -0.21 | +1.16 | +1.44 |

Table 61: Improvement of the memory power accuracy and precision considering the read-to-write ratio

| | Improvement | | | |
|---|---|---|---|---|
| | mean | median | standard deviation | variance |
| $PC1.1.2$ | -0.04 | +0.52 | +0.24 | +0.62 |
| $PC1.1.4$ | +1.01 | +1.08 | +0.12 | +0.26 |
| $PC2.1.4$ | -0.09 | +0.16 | -0.10 | -0.20 |
| $PC2.1.8$ | +0.59 | +1.03 | +0.07 | +0.11 |
| $PC2.2.8$ | +0.6 | +0.92 | +0.02 | +0.04 |
| **Mean** | **+0.41** | **+0.74** | **+0.07** | **+0.17** |

[365] Median: value in the middle when values are sorted in an ascending order
[366] Standard deviation: $\sigma$, root-mean-square (RMS) value from a set of *absolute differences* [W]
[367] Precision (variance): $\sigma^2$, closeness / variability of all data values

We demonstrate our memory power values exclusively considering the utilization levels and found that a precise simulation requires additional data about the memory instructions because the measured power values differ in dependence on the benchmark when tracing the same utilization level. In fact, if we do not consider the read-to-write ratio, the simulation results are definitely inadequate. Therefore, we always consider the workload scenario in our simulation to cover the read-to-write ratio of the memory module. Herein, we have to estimate the significant read-to-write ratio of each workload scenario (e.g. processor-bounded, memory-bounded, or I/O-bounded), which are based upon empirical studies by a statistical approximation.

***Memory Temperature Evaluation***

Furthermore, we simulate the memory temperature on the basis of the memory power consumption (directly dependent upon the utilization levels) and the technical specification. If a memory is in a steady state, such as a constant utilization level, the temperature will continuously increase, known as self-heating due to the power dissipation[368], which we observe closely to some individual phases in our measurements. Usually, the memory temperature does not increase as fast as the effect of an increasing utilization level will have. We specify a higher slope of the temperature increase in comparison to the decrease, which rely upon the delta between the previous and actual memory power. Therefore, we distinguish between a steady state, an increasing, and a decreasing temperature development. We specify all temperature-based methods by a time delay because of the inertia of the thermal development, as specified in Section 5. We neglect short-term peaks (less than one second) of the power consumption, which do not influence the memory temperature.

Again, we analyze the $4GB$ memory module of *Micron* ($LV\ DDR3-SDRAM$, see Table 47), which we exemplarily assemble together with two enabled *Intel Xeon E5-2650v2* processors in our system under test. Figure 115 graphically presents the results of our memory temperature simulation (dash-dotted red line) in the middle of the figure, which we evaluate according to our measurements (solid blue line) at $(PM2.1.4)$. In this example, the utilization level is in the range of $[22,29]\%$ of those results in the steady memory power states because of our cluster method (threefold division). We are aware of the read-to-write ratio, and thus we observe a temperature increase during the database operations and the read-to-write-phases[369]. The simulated temperature increases from $34.13°C$ up to $37.04°C$ during the database operations at the time $T=[1.089*10^4, 3.289*10^4]s$ and decreases slowly to $34.37°C$ at the time $T=[3.289*10^4, 5.439*10^4]s$.

[368] Self-heating: thermal response, predominantly specified by the power dissipation and thermal resistance

[369] Database operations and the read-to-write-phases: see Figure 108

Herein, we overestimate the temperature more than the rest of the *PassMark Memory* benchmarks probably because of the continuous changing read-to-write ratio. The simulated memory temperature increases at the times $T = [5.439 * 10^4, 7.339 * 10^4]s$, $T = [9.139 * 10^4, 1.044 * 10^5]s$, and $T = [1.344 * 10^5, 1.584 * 10^5]s$ due to the utilization level changes, which we consider in our temperature method.

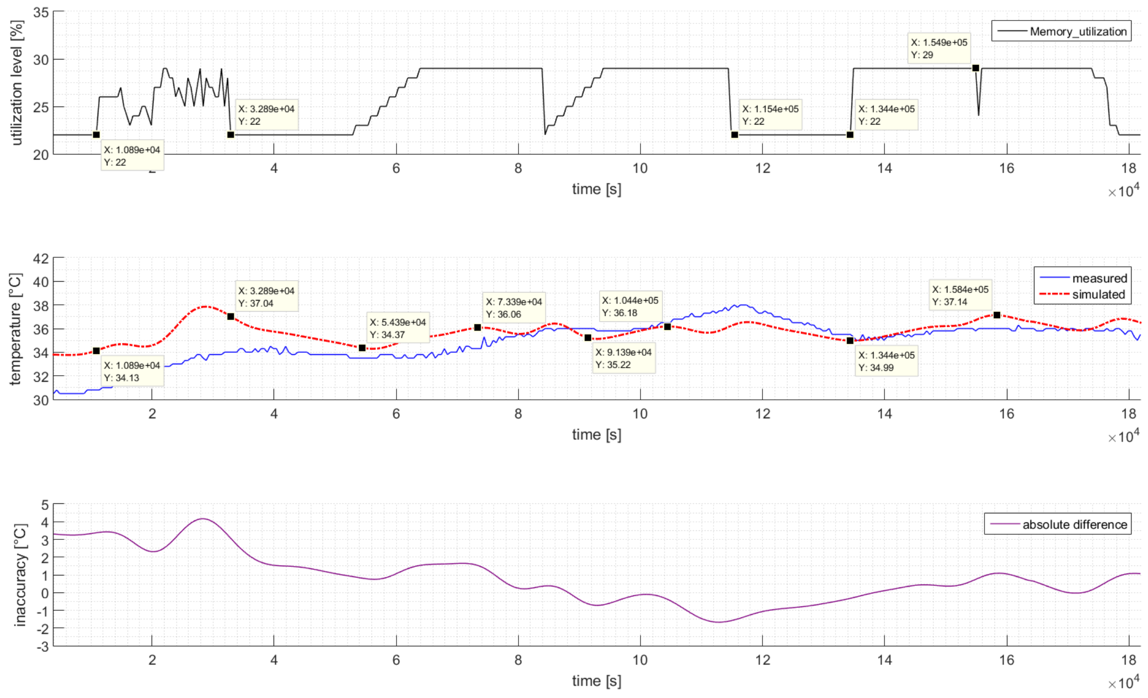


**Figure 115: Memory temperature accuracy $(PM2.1.4)$**

In this example, we observe an *absolute* temperature difference (in the mean) by approximately $+0.82°C$ and a *relative difference* by around $+2.5\%$ inaccuracy in relation to our measurements during the entire benchmark. Figure 116 presents the memory temperature accuracy by the related normalized probability function (histogram) on the basis of the *absolute differences* in degree Celsius. We overestimate the memory temperature and observe a median at $+0.67°C$, a standard deviation by $+1.41°C$, and a variance of $+1.98°C$.

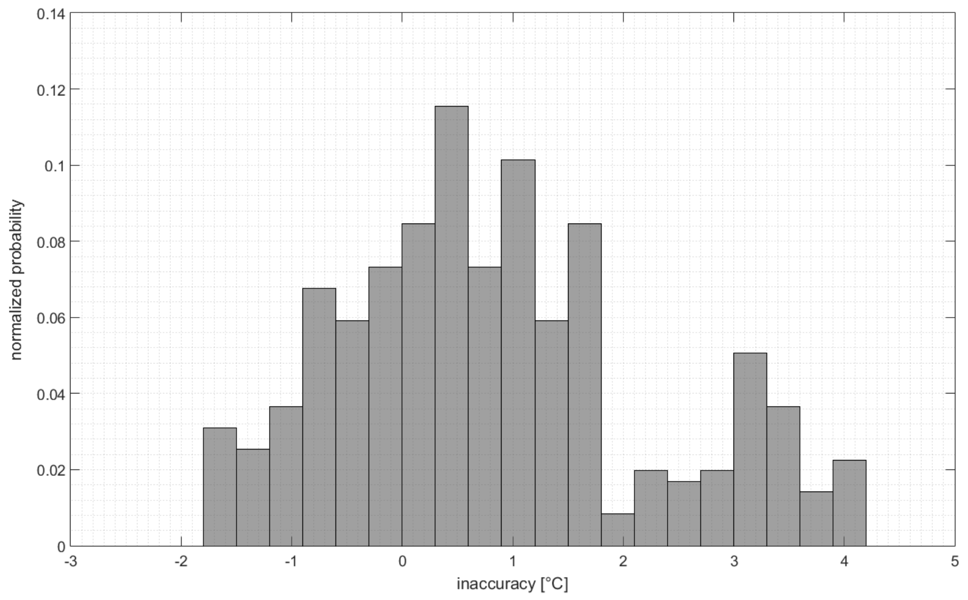


**Figure 116: Memory temperature accuracy – a histogram (normalized probability)** $(\boldsymbol{PM2.1.4})$

We observe a similar thermal development at the *PassMark CPU* benchmark $(PC2.1.4)$ because of the nearly steady utilization levels in the first third, as shown in the middle graph of Figure 117. We simulate the temperature increase at the time $T = [4167, 5.767 * 10^4]s$ from $33.79°C$ up to $34.37°C$, which does not increase as fast as the measured temperatures from $30.8°C$ up to $34.37°C$. A higher and faster temperature increase (up to $36.85°C$) occurs in the short-term at $T = [5.767 * 10^4, 8.067 * 10^4]s$ while calculating the prime numbers, which afterwards decreases to its previous value. In contrast, the measured temperature continuously increases during the time and has a low degree of dependency concerning the read-to-write ratio or utilization level, which is shown at the time between $T = [1.407 * 10^5, 1.597 * 10^5]s$. In our memory temperature simulation, we consider the power consumption on the basis of the read-to-write ratio and utilization level, which results in a higher temperature increase at the same time. We observe a temperature increase up to $38.56°C$ when the utilization level changes from $23\%$ up to $49\%$, but considering a time delay in respect to the inertia. Our simulation does not avoid the utilization-based gap at $t = 1.617 * 10^5 s$ because the low utilization lasts about several seconds at $T = [1.597 * 10^5, 1.642 * 10^5]s$. The effect is a wide range of the temperature decrease, with nearly $4°C$ between the two read-to-write intensive phases. We observe an *absolute* temperature difference (mean) by approximately $+0.8°C$ and a *relative difference* by around $+2.4\%$.

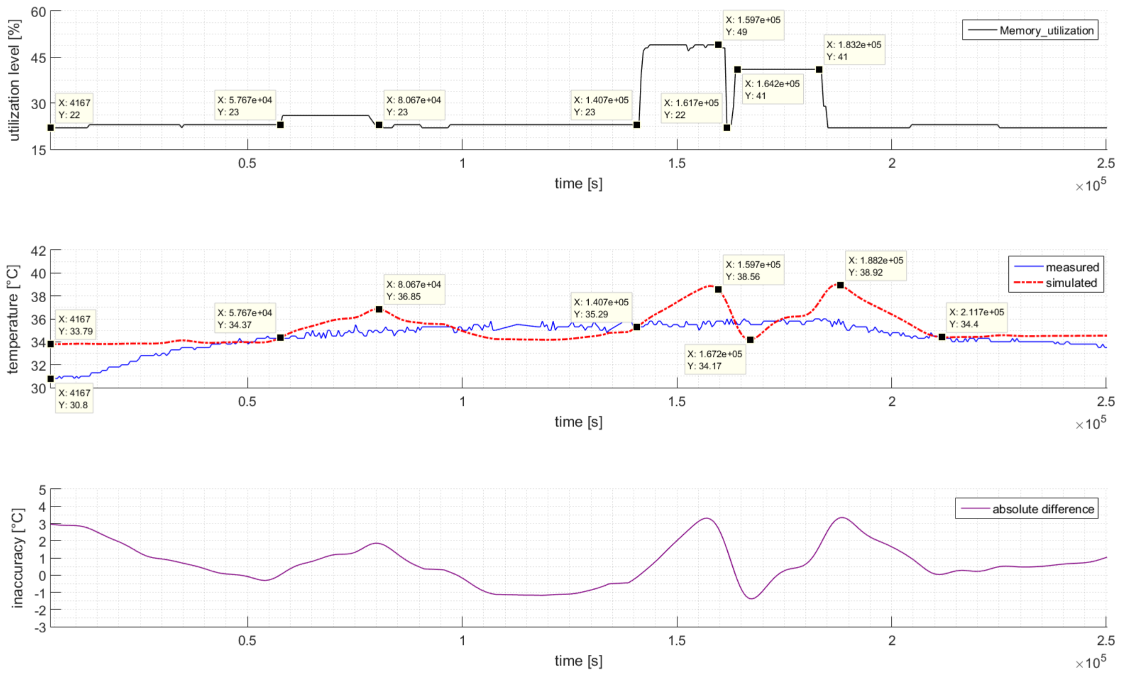


**Figure 117: Memory temperature accuracy $(PC2.1.4)$**

Table 62 lists the inaccuracy of our memory temperature simulation, their *absolute* as well as *relative differences* (in the mean) between the simulated and measured values considering various benchmarks. We overestimate the memory temperatures that differ in the range between $[\pm 0, +10]\%$ and observe that our simulation is more accurate considering modules with higher capacities. If we compare the results of the following benchmark pairs $(SP1.2.8, SP2.2.16)$, $(PC1.1.2, PC2.1.4)$, and $(PC1.1.4, PC2.2.8)$ that contain a second processor and the doubled memory amount, we observe a higher inaccuracy, despite having the same conditions of the memory temperature simulation. We assume that the additional processor may influence the thermal measurement of our memory modules, which we do not observe at the *PassMark Memory* benchmark. We overestimate the memory temperatures around $+1.2°C$, which is an approximate inaccuracy of $+4\%$.

Table 62: Memory temperature accuracy in comparison to the measurements

| | | Inaccuracy | |
|---|---|---|---|
| | | *Absolute difference* [°C] (mean) | *Relative difference* [%] (mean) |
| ***SPECpower*** | $SP1.2.8$ | *+0.9* | *+3.9* |
| | $SP2.2.16$ | *+2.58* | *+10* |
| ***PassMark CPU*** | $PC1.1.2$ | *+1.29* | *+3.9* |
| | $PC1.1.4$ | *+0.25* | *+1.1* |
| | $PC1.2.4$ | *+0.51* | *+1.8* |
| | $PC2.1.4$ | *+0.8* | *+2.4* |
| | $PC2.2.8$ | *+2.02* | *+6.3* |
| ***PassMark Memory*** | $PM1.1.2$ | *+1.82* | *+5.5* |
| | $PM1.1.4$ | *+0.8* | *+3* |
| | $PM1.2.4$ | *+2.24* | *+7.1* |
| | $PM2.1.4$ | *+0.82* | *+2.5* |
| | $PM2.2.8$ | *-0.01* | *+0.4* |

***Processor Power Evaluation***

In this section, we focus on the processor power and temperature, which are affected by the multiple processor cores and their dynamically changing frequencies. In contrast to the memory modules, we reproduce the specific utilization levels – or usage – in the range $u_{CS} = [0,100]\%$ in about $10\%$ discrete steps using the *SPECpower* benchmark.

In the beginning of this section, we justify our decision on the adequate measurement tools concerning their resolution to sufficiently trace the utilization levels and frequencies. Afterwards, we analyze the deviations of the thread-specific frequencies[370] of the respective processor and check whether the frequencies may improve the accuracy and precision of our processor model. Additionally, we review the measurement traces, analyze the mean values of the thread-specific and processor-specific frequencies, and test if we can neglect the thread-specific frequencies to speed up our simulation. The frequencies depend upon the utilization levels and therefore we show how the thread-specific utilization levels vary from the target throughput in the *SPECpower* benchmark and analyze their relevance when tracing the processor-specific utilization levels (as mean values) at the same time, which may accelerate the processor simulation.

[370] Thread-specific frequencies: consider the frequency of each processor core and related hardware thread

In the following section, we present the simplification of our measurement traces and simulation results, especially of the *SPECpower* benchmark and *MemTest86* to improve the readability of the results. According to the decision on what measurement tools and resolution we rely on, we present our simulation results regarding the processor power and temperature.

As stated in the concept chapter, we consider a resolution of one second in our simulation framework, which is our major assumption and abstraction when designing the processor model and simulating the aspects, for example. Concerning the experimental environment, three measurement tools are available, one with a $1000ms$ resolution, another with $100ms$ resolution, and the next one with $50ms$ resolution, see Table 49. First, we evaluate whether the 1-second resolution tool provides sufficient precision, and finally decide on the specific measurement tool of our processor evaluation that we rely on.

In order to analyze the processor frequency and the suitable tools, we start with the *PassMark CPU* benchmark[371] and exemplarily show the measurement results, beginning with the lowest resolutions of $100ms$ and $50ms$. As a typical illustration, Figure 118 presents the processor[372] frequency recorded by the *Intel Power Gadget* (top of the graph) and *Intel Power Thermal Utility* (middle of the graph) executing the *PassMark CPU* benchmark. At the bottom of the graph, we place both signals on top of each other so that the differences between them can be determined, which is our basis for analyzing the inaccuracy in relation to the sampling rate for the scope of our measurement infrastructure. We have to be aware that the frequencies are only instantaneous samples, not averages.

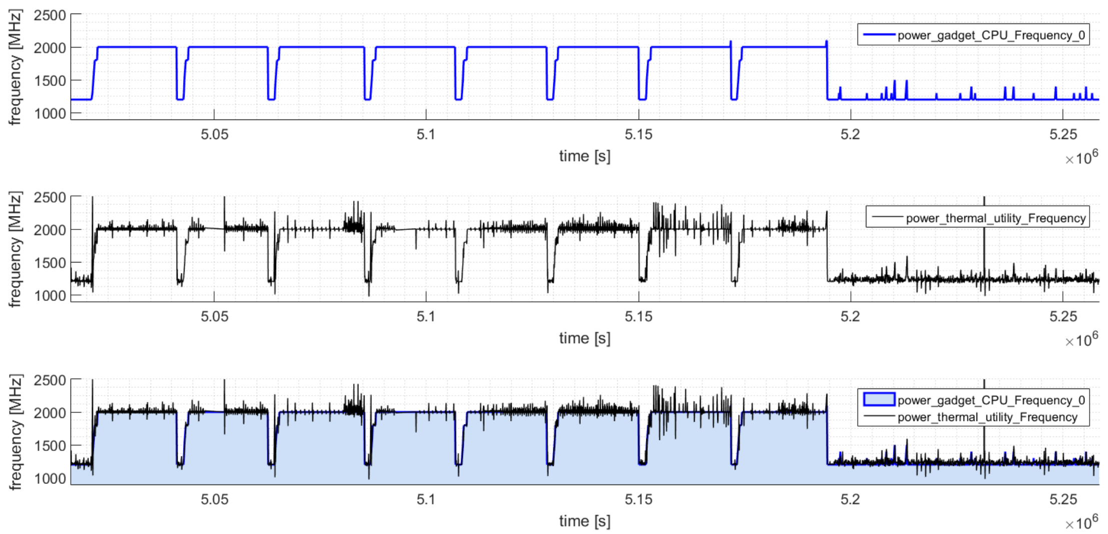


**Figure 118: *PassMark CPU* $(PC1.1.2)$ – processor frequency analysis (*Intel Power Gadget* vs. *Intel Power Thermal Utility*)**

[371] *PassMark CPU*: It is more transparent and easier to understand than the *SPECpower* illustration.
[372] Processor: first processor, called 'CPU0'

We calculate the integral of the areas under both curves considering the trapezoidal numerical integration (*trapz*) method of MATLAB that approximates the signals because we cannot define a specific $y = f(x)$ function. We define the *Intel Power Thermal Utility* signal as our basis because of the more accurate measurement results ($50ms$ resolution) in comparison to the remaining signal. Figure 119 and Figure 120 show an extract of some recordings, the first at the full utilization level and the second when the system is idle. The high-resolution data of the *Intel Power Thermal Utility* constantly vary in contrast to the signal of the *Intel Power Gadget*. Ideally, the processor frequency is steady at a particular value in both figures.

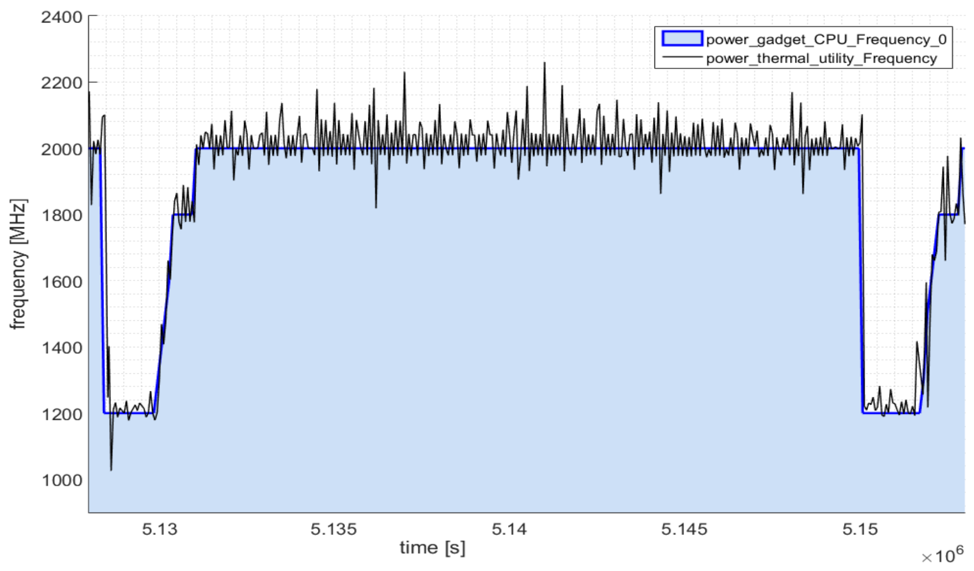


**Figure 119: Frequency analysis $(PC1.1.2)$ – 100% utilization level (*Intel Power Gadget* vs. *Intel Power Thermal Utility*)**

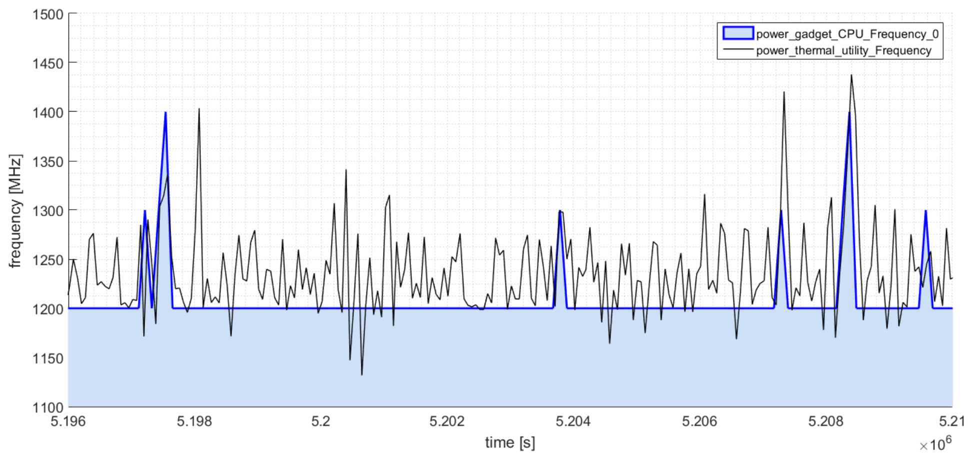


**Figure 120: Frequency analysis $(PC1.1.2)$ – 0% utilization level (*Intel Power Gadget* vs. *Intel Power Thermal Utility*)**

The frequencies change so often because of the power management of the processor/system, OS dependencies, or page faults in the idle case. We observe that the integrals of the areas under both curves are closely identical: e.g., we calculate between the *Intel Power Thermal*

*Utility* and the *Intel Power Gadget* signals a relative error in median of nearly $+0.68\%$ at the *PassMark CPU* benchmark and approximately $+1.48\%$ at the *SPECpower* benchmark. Table 63 lists the relative errors as the statistical representation regarding the *PassMark CPU* and *SPECpower* that Figure 121 graphically presents. The median value increases along the length of time that a benchmark requires and the *PassMark CPU* errors are skewed more than the *SPECpower* values, as shown in Figure 121.

**Table 63: Frequency inaccuracy (resolution inaccuracy: *Intel Power Gadget* vs. *Intel Power Thermal Utility*) – relative error (numerical) of *PassMark CPU* and *SPECpower***

| Resolution inaccuracy (*trapz* of frequency) | *PassMark CPU* | *SPECpower* |
|---|---|---|
| **Mean time [min]** | 4.04 | 76.32 |
| **Upper adjacent [%]** | +1.18 | +5.40 |
| **75th percentile [%]** | +1.04 | +2.91 |
| **Median [%]** | **+0.68** | **+1.48** |
| **25th percentile [%]** | +0.55 | +0.15 |
| **Lower adjacent [%]** | +0.30 | -2.41 |

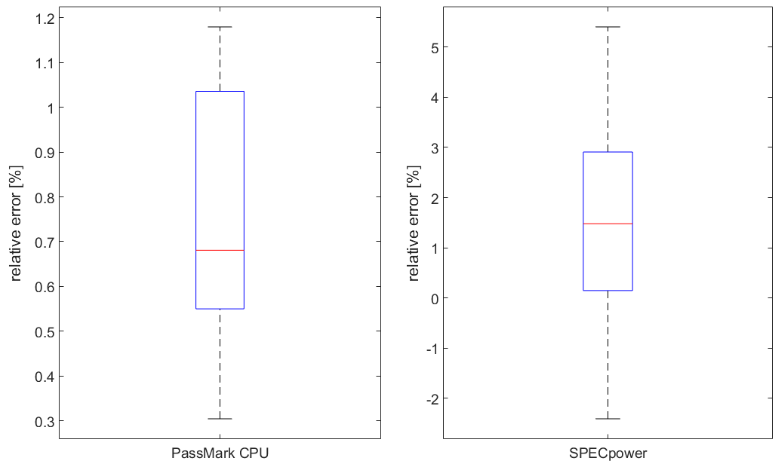


**Figure 121: Frequency inaccuracy – graphical representation of the relative error of *PassMark CPU* and *SPECpower* (resolution inaccuracy: *Intel Power Gadget* vs. *Intel Power Thermal Utility*)**

Thus, we assume that the *Intel Power Gadget* is sufficiently precise to trace the processor frequency because the error rate is tolerable in the entire experimental analyses. Moreover, we analyze the *HW Monitor Pro* with a sample time on a 1-second basis and check whether we can use the tool instead of the *Intel Power Gadget*, whereby the *HW Monitor Pro* has the identical resolution such as our simulation model.

The analysis procedure concerning the processor frequency and tool resolution itself is the same: we calculate the integral of the areas under both curves of the *HW Monitor Pro* and the *Intel Power Gadget*, whereby we specify the *Intel Power Gadget* as the new basis of our calculation. Figure 122 presents the processor frequency[373] recorded by the *Intel Power Gadget* (top of the graph) and *HW Monitor Pro* (middle of the graph) running the *PassMark CPU* benchmark.

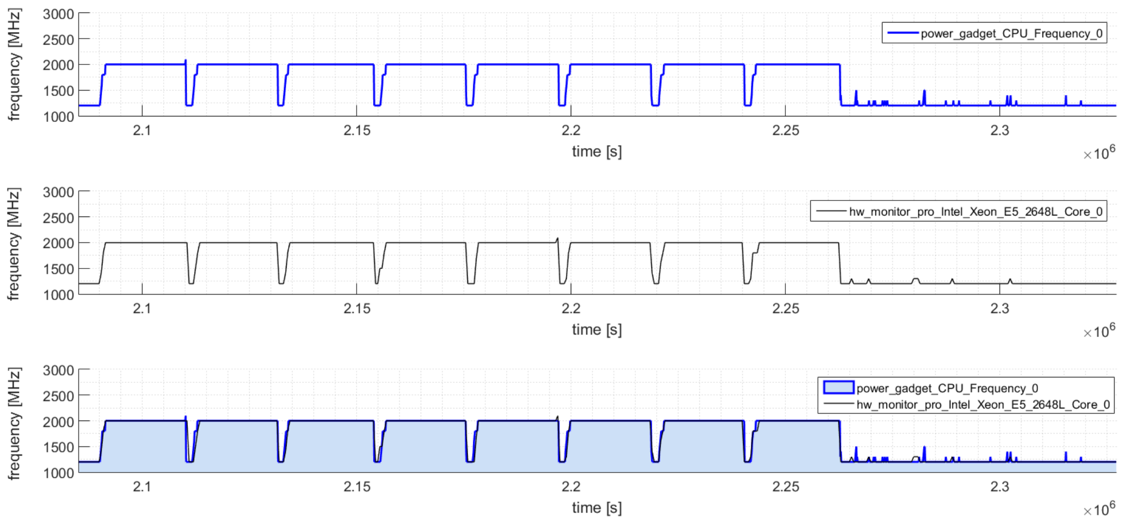


**Figure 122: *PassMark CPU* $(PC1.1.4)$ – processor frequency analysis (*Intel Power Gadget vs. HW Monitor Pro*)**

We observe that the curves of the *Intel Power Gadget* and the *HW Monitor Pro* are almost identical, especially when executing the *PassMark CPU* benchmark. As a rule, we observe that the processor frequency is always at the highest possible level at $2000MHz$ during the active phases of the micro-benchmarks in the *PassMark CPU*.

Figure 122 shows an ideal representation of our measurement tool. In contrast, we observe that the signal of the *HW Monitor Pro* performs out of sync in a couple of traces, as exemplarily shown in Figure 123. Both curves are nearly identical between $T = [5.041 * 10^6, 5.044 * 10^6]s$ at a frequency of $2000MHz$. The frequency levels get out of sync, such as at $T = [5.058 * 10^6, 5.066 * 10^6]s$, and continuously begin to move apart from each other, which is especially observable at the time $T = [5.098 * 10^6, 5.11 * 10^6]s$. In some *PassMark CPU* runs, the effects of time delay only occur intermittently after executing the third or fourth micro-benchmark. Indeed, if we analyze the measurement results gained from the *HW Monitor Pro* in comparison with the other tools, we have to consider the occurring time problems. In the beginning of the frequency analysis in Table 67 and Table 68, we neglect the time differences of the measurement tools because the *SPECpower* benchmark interrupts the target throughput by some seconds.

[373] Processor frequency: The first processor, called ‘CPU0’, and the first core, called ‘core 0’, both belong together.

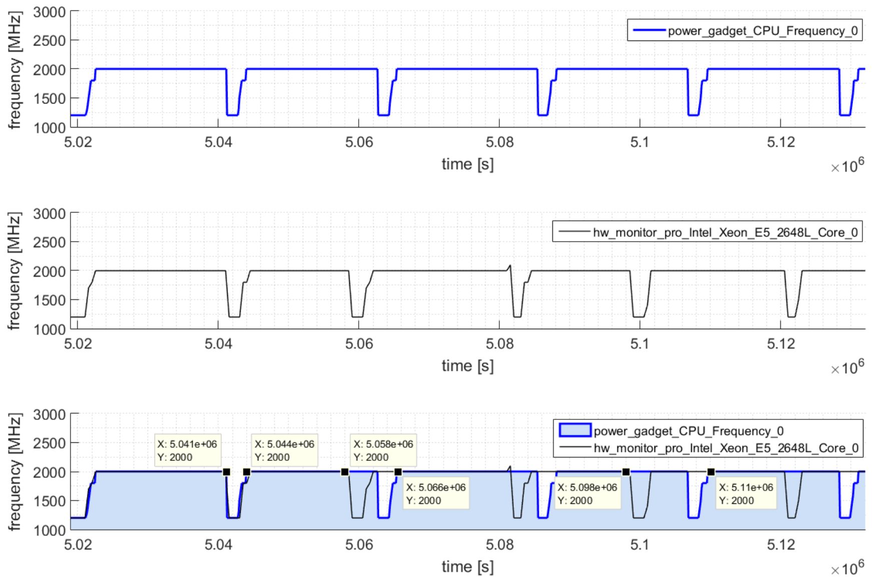


**Figure 123: *PassMark CPU* $(PC1.1.2)$ – processor frequency analysis (*Intel Power Gadget vs. HW Monitor Pro*)**

Additionally, we observe that the *HW Monitor Pro* does not completely trace all benchmarks before the end of their term. In general, we want to avoid such synchronization errors[374] to be more precise in our evaluation.

In our analysis, between the *HW Monitor Pro* and the *Intel Power Gadget* signals we calculate a relative error in median of approximately $-0.22\%$ at the *PassMark CPU* benchmark and nearly $-0.08\%$ at the *SPECpower* benchmark. Table 64 lists the relative errors as the statistical representation regarding the *PassMark CPU* and *SPECpower* that Figure 124 graphically presents.

**Table 64: Frequency inaccuracy (resolution inaccuracy: *Intel Power Gadget* vs. *HW Monitor Pro*) – relative error (numerical) of *PassMark CPU* and *SPECpower***

| **Resolution inaccuracy (*trapz* of frequency)** | ***PassMark CPU*** | ***SPECpower*** |
|---|---|---|
| **Mean time [min]** | 4.04 | 76.32 |
| **Upper adjacent [%]** | -0.14 | +7.86 |
| **75th percentile [%]** | -0.18 | +2.91 |
| **Median [%]** | **-0.22** | **-0.08** |
| **25th percentile [%]** | -2.93 | -2.50 |
| **Lower adjacent [%]** | -6.50 | -9.71 |

[374] Synchronization errors: associated by the first quartile or lower adjacent values in Table 64

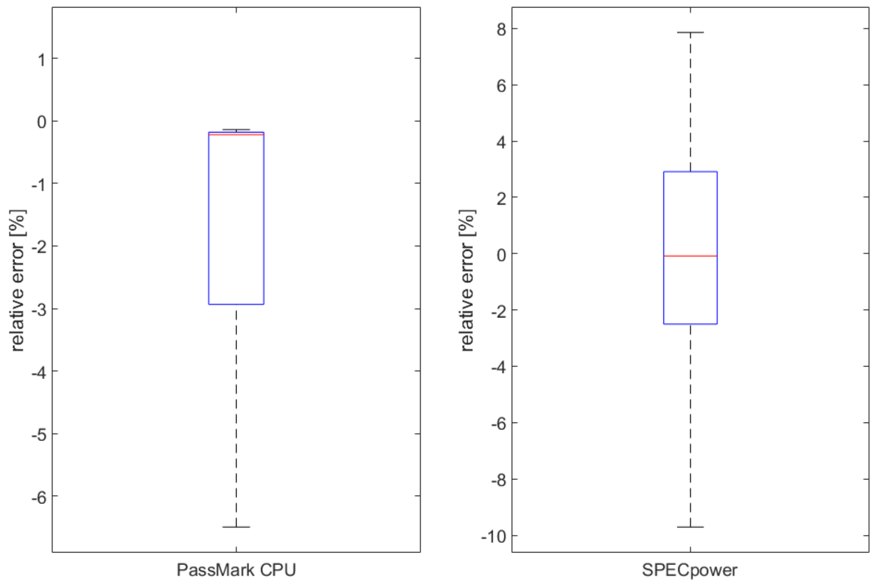


**Figure 124: Frequency inaccuracy – graphical representation of the relative error of *PassMark CPU* and *SPECpower* (resolution inaccuracy: *Intel Power Gadget* vs. *HW Monitor Pro*)**

We assume that the error rate is tolerable in entirely experimental analyses because the configurations and characteristics have a greater impact on the models. The tool with a sample time on a 1-second basis provides sufficient precision. Nevertheless, we choose the *Intel Power Gadget* for our evaluation that provides the same time stamps and is more precise than the *HW Monitor Pro.* Consequently, we trace the processor frequency by the *Intel Power Gadget* and consider their values in our evaluation with a $100ms$ resolution. We study the identical utilization trace when we simulate the processor, which works at a resolution of $1s$. Therefore, we compare the experimental trace and our model on a 1-second basis.

The authors of [Han 2007] studied the effects of the sampling rate considering a range of sampling interval sizes at the *SPEC CPU2000*[375] benchmark. The authors argued that a higher sample rate detects short power peaks more often than a lower rate, which may lead to another power management decision. In contrast, we neglect the peaks and a resolution lower than $1s$ because we do not optimize the processor power on a cycle-by-cycle basis.

The authors of [TDM 2011, MAC et al. 2011] propose a power management strategy that schedules the various frequencies of a multi-core processor. Consequently, we investigate the potential improvement on the thread-specific frequencies of a processor. In our analysis, we exemplarily trace the thread-specific frequencies by the *HW Monitor Pro* while executing the *SPECpower* benchmark ($SP1.2.8$). We statistically analyze the frequencies of all hardware threads[376] that occur in the benchmark phases presented as target throughput in percentage, as shown in Table 65 and Table 66. We observe extremely small deviations from the hardware

[375] *SPEC CPU2000*: https://www.spec.org/cpu2006/

[376] Hardware thread: The SUT processor consists of eight cores with hyper-threading technology. Consequently, the processor provides 16 hardware threads.

threads, such as nearly identical frequencies $(\pm 8MHz)$, which is recognizable by the interquartile range[377] (IQR). We additionally provide an overview of the detailed frequencies of all hardware threads in the Appendix A3f. We assume that the processor architecture is responsible of nearly the same frequencies that result on the least common multiple (LCM) and consider the mean value instead. We similarly observe the identical thread-specific frequencies of the *Intel Xeon* architecture[378]. The thread-specific frequencies and their deviations do not significantly affect the processor power consumption.

**Table 65: Statistical representation of the thread-specific frequencies [MHz] of the processor at target throughput $(\boldsymbol{calibration}, \mathbf{100\%} - \mathbf{60\%})$ in $(\boldsymbol{SP1.2.8})$**

| Thread-specific frequencies [MHz] | Target throughput [%] | | | | | |
|---|---|---|---|---|---|---|
| | Calibration | 100 | 90 | 80 | 70 | 60 |
| **Upper adjacent** | 1997 | 1988 | 1813 | 1688 | 1540 | 1427 |
| **75th percentile** | 1996 | 1987 | 1813 | 1687 | 1538 | 1423 |
| **Median** | **1995** | **1987** | **1811** | **1687** | **1535** | **1421** |
| **25th percentile** | 1994 | 1987 | 1811 | 1680 | 1530 | 1419 |
| **Lower adjacent** | 1993 | 1987 | 1811 | 1677 | 1527 | 1416 |
| **Interquartile range** | **2** | **0** | **2** | **7** | **8** | **4** |
| | | | | | | |
| **Mean** | **1995** | **1987** | **1812** | **1684** | **1534** | **1421** |

**Table 66: Statistical representation of the thread-specific frequencies [MHz] of the processor at target throughput $(\mathbf{50\%} - \mathbf{10\%}, \boldsymbol{idle})$ in $(\boldsymbol{SP1.2.8})$**

| Thread-specific frequencies [MHz] | Target throughput [%] | | | | | |
|---|---|---|---|---|---|---|
| | 50 | 40 | 30 | 20 | 10 | Idle |
| **Upper adjacent** | 1337 | 1267 | 1226 | 1205 | 1200 | 1200 |
| **75th percentile** | 1334 | 1265 | 1225 | 1204 | 1200 | 1200 |
| **Median** | **1333** | **1262** | **1222** | **1204** | **1200** | **1200** |
| **25th percentile** | 1332 | 1260 | 1221 | 1203 | 1200 | 1200 |
| **Lower adjacent** | 1332 | 1258 | 1219 | 1203 | 1200 | 1200 |
| **Interquartile range** | **2** | **5** | **4** | **1** | **0** | **0** |
| | | | | | | |
| **Mean** | **1333** | **1262** | **1223** | **1204** | **1200** | **1200** |

If we compare the mean values of the thread-specific frequencies gained from the *HW Monitor Pro* with the processor-specific frequencies of the *Intel Power Gadget*, we observe almost equal frequencies $(\pm 6MHz)$ at both tools of the various target throughputs, as shown in Table

[377] Interquartile range: 75th percentile minus 25th percentile, upper (third) quartile minus lower (first) quartile

[378] *Intel Xeon* architecture: *Intel Xeon* processor *E3/5/7*-xxxx *v1-5* family

67 and Table 68. In our accuracy analysis, we can consider both tools concerning the mean frequency at a specific target throughput. As an additional result, we neglect the thread-specific frequencies, because the values do not provide extra data that improve the accuracy or precision of our processor power model, and concentrate on the processor-specific frequencies gained by the *Intel Power Gadget*, which saves time and reduces the effort.

Table 67: Mean frequency [MHz] at target throughput $(\boldsymbol{calibration}, \mathbf{100\%} - \mathbf{60\%})$ in $(\boldsymbol{SP}\mathbf{1.2.8})$

| | Target throughput [%] | | | | | |
|---|---|---|---|---|---|---|
| **Mean frequency [MHz]** | **Calibration** | **100** | **90** | **80** | **70** | **60** |
| ***HW Monitor Pro*** | 1995 | 1987 | 1812 | 1684 | 1534 | 1421 |
| ***Intel Power Gadget*** | 1998 | 1991 | 1818 | 1690 | 1540 | 1421 |
| ***Absolute difference*** | **-3** | **-4** | **-6** | **-6** | **-6** | **0** |

Table 68: Mean frequency [MHz] at target throughput $(\mathbf{50\%} - \mathbf{10\%}, \boldsymbol{idle})$ in $(\boldsymbol{SP}\mathbf{1.2.8})$

| | Target throughput [%] | | | | | |
|---|---|---|---|---|---|---|
| **Mean frequency [MHz]** | **50** | **40** | **30** | **20** | **10** | **Idle** |
| ***HW Monitor Pro*** | 1333 | 1262 | 1223 | 1204 | 1200 | 1200 |
| ***Intel Power Gadget*** | 1333 | 1261 | 1222 | 1204 | 1200 | 1200 |
| ***Absolute difference*** | **0** | **+1** | **+1** | **0** | **0** | **0** |

In particular, the processor frequencies result from the actual utilization level, which both are the major parameters in our simulation model concerning the processor power consumption and related temperature. We assume that the thread-specific utilization levels depend upon the target throughput, as it has already been true to the thread-specific frequencies, which we further analyze considering the traces of the *HW Monitor Pro*. We exemplarily present the statistical results of the thread-specific utilization levels in Table 69 and Table 70. We observe that the thread-specific utilization levels vary less from the calibration phases up to a target throughput of $60\%$ than at the subsequent target throughput of $(50\% - 10\%, idle)$, presented by the interquartile range. Mostly, all thread-specific utilization levels evenly rely upon the target throughput of the benchmark, and we discover that all threads are more involved permanently at the beginning of the benchmark achieving the target throughput, but some of them become unused at the end. Furthermore, we observe that the benchmark utilizes all threads in the same way with one exception: the *zero* thread[379]. Accordingly, we observe a utilization level of the *zero* thread at $34.4\%$ in the mean during the idle time (specified as an outlier), which is nearly three times the mean value of approximately $11.73\%$. In the remaining target throughputs, the *zero* thread is nearby the lower adjacent. We assume that the *zero* thread will behave like the residual threads, especially in the idle case when the thread does not execute additional tasks, such as executing the benchmark, writing data onto

[379] *Zero* thread: first processor core and related thread

discs, or administrating tasks. When we consider the fact that the frequencies vary only $(\pm 8MHz)$ and at the same time, the utilization levels deviate between $[0-15]\%$ of the particular target throughput, we can neglect the thread-specific utilization levels that are irrelevant and consider the mean value instead.

**Table 69: Statistical representation of the thread-specific utilization levels [%] of the processor at target throughput $(\boldsymbol{calibration}, \mathbf{100\%-60\%})$ in $(\boldsymbol{SP1.2.8})$**

| Thread-specific utilization levels [%] | Target throughput [%] | | | | | |
|---|---|---|---|---|---|---|
| | Calibration | 100 | 90 | 80 | 70 | 60 |
| **Upper adjacent** | 99.66 | 97.14 | 86.14 | 77.23 | 68.59 | 60.26 |
| **75th percentile** | 99.56 | 96.86 | 84.70 | 76.08 | 65.93 | 59.00 |
| **Median** | **99.42** | **96.14** | **83.43** | **73.72** | **64.79** | **55.37** |
| **25th percentile** | 99.28 | 94.06 | 78.15 | 70.26 | 63.53 | 54.27 |
| **Lower adjacent** | 98.94 | 92.28 | 76.16 | 65.24 | 62.50 | 49.61 |
| **Interquartile range** | **0.28** | **2.80** | **6.55** | **5.82** | **2.40** | **4.73** |
| **Mean** | **99.4** | **95.03** | **81.08** | **72.17** | **64.04** | **56.08** |

**Table 70: Statistical representation of the thread-specific utilization levels [%] of the processor at target throughput $(\mathbf{50\%-10\%}, \boldsymbol{idle})$ in $(\boldsymbol{SP1.2.8})$**

| Thread-specific utilization levels [%] | Target throughput [%] | | | | | |
|---|---|---|---|---|---|---|
| | 50 | 40 | 30 | 20 | 10 | Idle |
| **Upper adjacent** | 54.69 | 46.22 | 36.18 | 31.02 | 13.56 | 23.40 |
| **75th percentile** | 53.18 | 42.66 | 34.95 | 22.68 | 12.28 | 13.85 |
| **Median** | **43.83** | **35.81** | **27.68** | **18.45** | **9.49** | **10.80** |
| **25th percentile** | 42.21 | 31.13 | 20.55 | 13.97 | 6.54 | 6.15 |
| **Lower adjacent** | 40.10 | 30.19 | 19.21 | 12.77 | 5.25 | 0 |
| **Interquartile range** | **10.97** | **11.53** | **14.40** | **8.71** | **5.74** | **7.70** |
| **Mean** | **46.91** | **36.83** | **27.73** | **19.08** | **10.49** | **11.73** |

We calculate the mean utilization level of all available hardware threads considering the *HW Monitor Pro*, which requires additional computational and analysis effort. We assume that the processor-specific utilization levels of the *Intel Power Thermal Utility* are equally suitable for our analysis and evaluation of the accuracy and precision, but can be accessed easier. Therefore, we compare the mean utilization level of the entire processor considering both measurement tools applied on the same benchmark. We observe that the *Intel Power Thermal Utility* has an *absolute difference* in the mean of approximately $+2.2\%$ in comparison to the thread-specific utilization levels of the *HW Monitor Pro*, as shown in Table 71 and Table 72. The idle case builds an exception regarding the *absolute difference*, which varies from nearly

$-7.33\%$. We suggest that the measurement traces of the *Intel Power Thermal Utility* may be more precise in comparison to the *HW Monitor Pro*, because of the same manufacturer when analyzing the processor.

**Table 71: Mean utilization level [%] at target throughput $(calibration, 100\% - 60\%)$ in $(SP1.2.8)$**

| | Target throughput [%] | | | | | |
|---|---|---|---|---|---|---|
| **Mean utilization levels [%]** | **Calibration** | **100** | **90** | **80** | **70** | **60** |
| ***Intel Power Thermal Utility*** | 99.73 | 96.12 | 84.17 | 75.89 | 67.94 | 60.17 |
| ***HW Monitor Pro*** | 99.4 | 95.03 | 81.08 | 72.17 | 64.04 | 56.08 |
| ***Absolute difference*** | **+0.33** | **+1.09** | **+3.09** | **+3.72** | **+3.90** | **+4.09** |

**Table 72: Mean utilization level [%] at target throughput $(50\% - 10\%, idle)$ in $(SP1.2.8)$**

| | Target throughput [%] | | | | | |
|---|---|---|---|---|---|---|
| **Mean utilization levels [%]** | **50** | **40** | **30** | **20** | **10** | **Idle** |
| ***Intel Power Thermal Utility*** | 50.75 | 40.75 | 31.57 | 22.38 | 13.40 | 4.4 |
| ***HW Monitor Pro*** | 46.91 | 36.83 | 27.73 | 19.08 | 10.49 | 11.73 |
| ***Absolute difference*** | **+3.84** | **+3.92** | **+3.84** | **+3.30** | **+2.91** | **-7.33** |

As a result, we consider the processor-specific utilization levels[380] of the *Intel Power Thermal Utility* instead of the thread-specific utilization levels of the *HW Monitor Pro* as the reference parameter of our simulation and evaluation. Figure 125 exemplarily shows the processor-specific utilization levels, given as a percentage, which we trace by the *Intel Power Thermal Utility* while executing the *SPECpower* benchmark $(SP1.2.8)$. In the figure, we see the three calibration phases between $T = [1.122 * 10^6, 2.039 * 10^6]s$ searching for the maximum throughput, which consequently fully utilize the processor by approximately $100\%$. Moreover, the graph shows the actual utilization level of the processor, which ideally consists of several discrete and steady $10\%$ steps concerning the target throughput beginning at $t = 2.039 * 10^6 s$ until $t = 5.151 * 10^6 s$ followed by three idle intervals. In our measurements, we automatically detect[381] the beginning and the end of each interval that we mark by a vertical solid line. We observe that a specific target throughput does not guarantee a steady utilization level, as shown in Figure 125 by the utilization levels that move up and down.

[380] Processor-specific utilization levels: entire processor frequency independent of the cores, common in industrial practice

[381] Automatically detect: implement a certain MATLAB function for evaluation, detect the rising and falling edge of the utilization levels

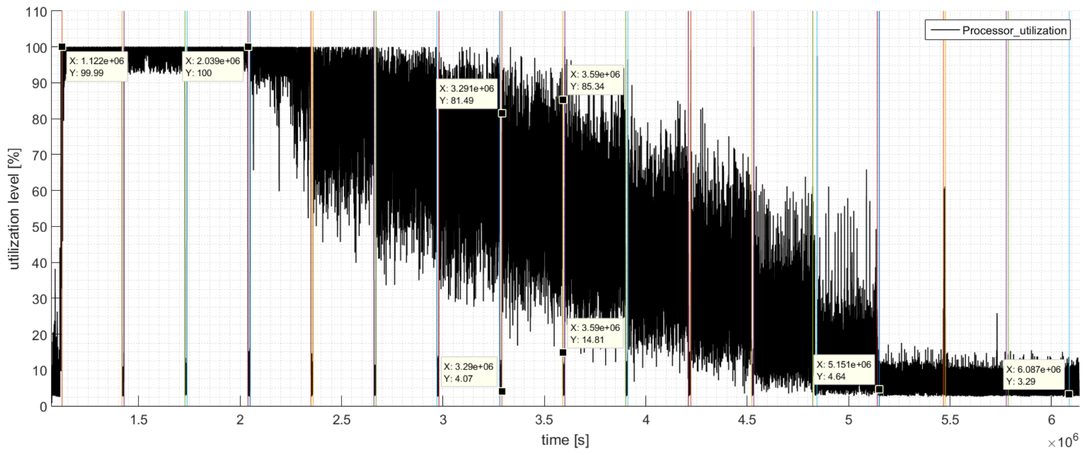


Figure 125: Processor utilization level $(SP1.2.8)$ measured by *Intel Power Thermal Utility*

As an exemplary illustration, we simplify the graph of Figure 125 to show an easier to read representation of the measurement results. Therefore, we use the times of each vertical solid line that indicate the interval length of each $10\%$ step (target throughput) to calculate the mean value of the actual utilization level. Herein, we consider a short transient phase by avoiding the first $200ms$ specified by an internal analysis to neglect the low utilization levels between the intervals, such as $4.07\%$ at $t = 3.29 * 10^6 s$ or $14.81\%$ at $t = 3.59 * 10^6 s$. Figure 126 shows the simplified representation of Figure 125 illustrating the same intervals marked by the identical vertical solid lines. Herein, we rename the horizontal axis to present the target throughput[382], which is implicitly represented by the time of the Figure 125. The vertical axis shows the actual utilization level in $[\%]$ calculated as a mean value of their particular interval, which we present as dashed-dotted black line labeled by their corresponding label. We expect a linear relation (ideal course) between the target throughput and the actual utilization level. In contrast, we observe that the mean values are lower than the target throughput from $100\%$ up to $70\%$ and, in fact, larger in the subsequent part of the benchmark. In the idle intervals starting at $t = 5.151 * 10^6 s$ and ending at $t = 6.087 * 10^6 s$, we observe the mean utilization level between $4.1\%$ and $4.9\%$, which shows that the processor is always busy and does not have a utilization level around $0\%$.

[382] Target throughput: defined as target loads at *SPECpower* benchmark

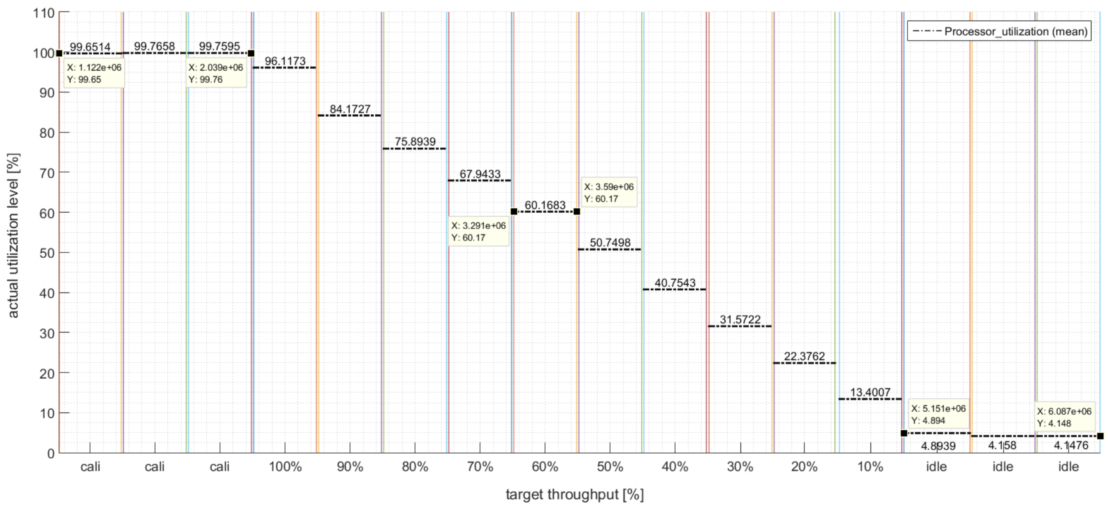


**Figure 126: Simplified representation of the processor utilization level $(SP1.2.8)$ as mean values**

In our evaluation, we consider the high-resolution and precision of the *Intel Power Thermal Utility* regarding the processor-specific utilization levels, shown in Figure 125, as the reference parameter of our simulation model, but present the results in the simplified and readable graph of their mean values. We consider the actual utilization levels of the complete processor that are approximately linearly dependent on target throughput, which we consider when simulating the performance-to-power ratio.

The key performance metric of the processor is the actual processor-specific utilization level that indicates the physical usage of the entire processor, usually expressed in percentages. The utilization level has the major impact on the processor power consumption and corresponding thermal development.

We exemplarily analyze the *Intel Xeon E5-2650v2* processor (see Table 47), executing various benchmarks to guarantee various utilization levels and simulate the power consumption. Therefore, we execute the benchmarks on the real server system and trace the processor-specific utilization level (by the *Intel Power Thermal Utility*), power consumption (by the *Intel Power Gadget*), and temperature (by the *Intel Power Gadget*). We avoid the characteristic changes in this section because of the limited measurement results of our system under test, but we further analyze the characteristic changes of the *Intel Xeon E5-26xx v2* processor *generation* and *family* in Section 7.3 considering the Fujitsu-specific measurements and vendor-based spreadsheets.

We exemplarily present the identical benchmarks[383], as described in the memory evaluation, but in consideration of the processor power and temperature. In the meantime, the processor is under-utilized while executing the *PassMark Memory* benchmark, but immediately fully

[383] Benchmarks: We trace all component-specific utilization levels and corresponding aspects executing the same benchmark.

utilized when the memory executes the read/write operations. In contrast, the *PassMark CPU* benchmark almost fully utilizes the processor nearly the whole time. We do not observe utilization levels in the range of $[20,80]\%$, thus we additionally consider the *PassMark MemTest86* and the *SPECpower* benchmark, which especially reproduce the certain processor-specific utilization levels.

Figure 127 exemplarily shows the utilization levels while executing the *PassMark Memory* benchmark $(PM1.1.4)$ in the upper graph in which the benchmark utilizes the processor below $20\%$ (in the mean $11\%$) at $T = [2.624 * 10^6, 2.768 * 10^6]s$ and fully utilizes the processor at $T = [2.768 * 10^6, 2.789 * 10^6]s$, which we similarly observe in the remaining $(PMx.y.z)$ benchmark runs. We measure the processor power consumption of approximately $23W$ at the lower utilization levels and around $50W$ at the higher utilization levels, as shown in the middle graph by a solid blue line. We present our simulation-based results of the processor power as a dashed-dotted red line and show the *absolute difference* in the mean (marked in purple) between the measurements and our simulation in the lower graph of the figure. We observe nearly identical power consumptions, but observe an extreme inaccuracy in the range of $[-12.4, +29.3]W$ in a situation when the utilization level increases from $17\%$ to $100\%$, decreases to $24\%$, and increases to $100\%$ between $t = 2.768 * 10^6 s$ and $t = 2.769 * 10^6 s$. We can avoid these situations when we continuously analyze the utilization levels and neglect the detected peaks. Nevertheless, we estimate the power consumption by a mean inaccuracy of approximately $+0.19W$, which is an $+0.6\%$ inaccuracy in relation to our measurements during the entire benchmark in this example.

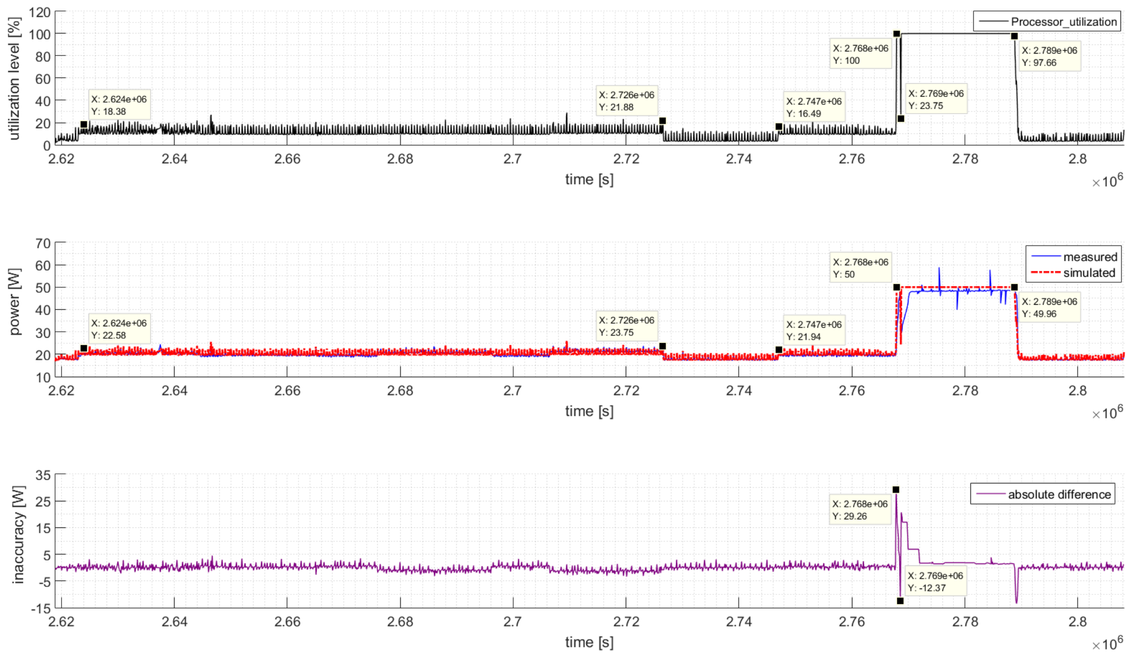


Figure 127: Processor power accuracy $(PM1.1.4)$

In the next section, we check whether we can trust the power consumption gained from the *PassMark Memory* benchmarks, which we use in our processor evaluation. For this purpose, we analyze the processor utilization levels in the *PassMark CPU* benchmark that concentrates on the processor utilization to detect the same processor power and show the plausibility of the measured and simulated values. The top graph in Figure 128 exemplarily shows the processor utilization levels (gained by the *Intel Power Thermal Utility*) of the identical server configuration used in $(PM1.1.4)$, but we execute the *PassMark CPU* benchmark $(PC1.1.4)$ instead. Herein, we observe that the benchmark fully utilizes the processor nearly the entire time at $T = [2.09 * 10^6, 2.262 * 10^6]s$ is interrupted by changing the particular micro-benchmark; an exception of the utilization levels builds the *single threaded* micro-benchmark in the end of the benchmark at $t = 2.263 * 10^6 s$, which utilizes the processor between $17\%$ and $24\%$, as exemplarily tagged in the figure. The graph in the middle illustrates the measured power consumption (by the *Intel Power Gadget*) presented by a solid blue line and the simulation-based shown as a dashed-dotted red line. We observe that the processor power consumption varies in the range of $[44.2, 66.8]W$ when we fully utilize the processor, but approximately observe a power consumption around $50W$. Indeed, we measure the following power consumptions when executing the particular micro-benchmarks:

| | | |
|---|---|---|
| • *Integer*: | $\sim 44.2W$ | at $T = [2.090 * 10^6, 2.109 * 10^6]s$ |
| • *Floating-point*: | $\sim 55.3W$ | at $T = [2.113 * 10^6, 2.131 * 10^6]s$ |
| • *Prime numbers*: | $\sim 44.2W$ | at $T = [2.134 * 10^6, 2.153 * 10^6]s$ |
| • *Encryption*: | $[48.2, 66.8]W$ | at $T = [2.199 * 10^6, 2.219 * 10^6]s$ |
| • *Sort strings*: | $\sim 54.4W$ | at $T = [2.242 * 10^6, 2.262 * 10^6]s$ |

We assume that the varying power consumptions depend upon the processor architecture and its integrated subsystems (see footnote 330). In contrast, we simulate a constant processor power of around $50W$, as illustrated at $t = 2.173 * 10^6 s$, which is only based upon the utilization levels, while neglecting the certain operations, and additionally analyze the accuracy. The lower graph of the figure presents the *absolute difference* in the mean (marked in purple) between the measurements and our simulation. We observe nearly identical power consumptions, but observe extreme inaccuracies, such as peaks in the range of $[-23.6, +26.9]W$ while switching the micro-benchmarks, because of the different resolutions between the measurement tools and our simulation framework. In fact, we estimate the power consumption by a mean inaccuracy of approximately $+0.02W$ in this example, which is an $+1.5\%$ inaccuracy in relation to our measurements of the entire benchmark.

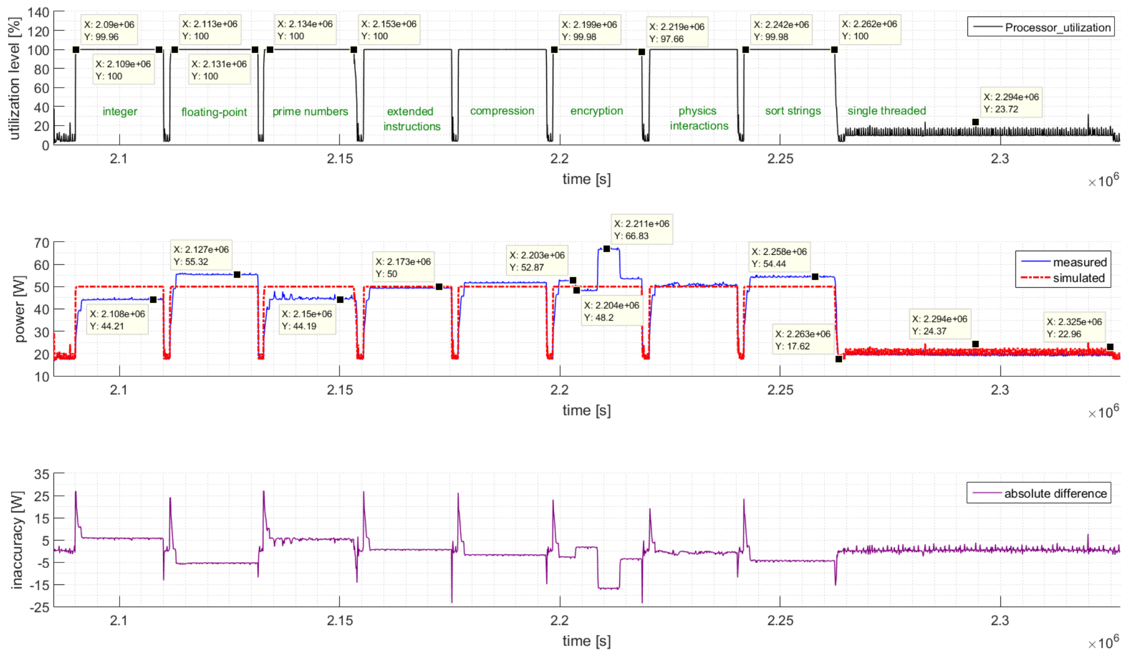


**Figure 128: Processor power accuracy $(PC1.1.4)$**

We do not observe utilization levels between $20\%$ and $80\%$ in the *PassMark Memory* or *PassMark CPU* benchmarks, but can argue that our simulation-based results are sufficiently precise. Furthermore, we analyze the results of the *SPECpower* benchmark in the following section, which explicitly adjusts the utilization levels of the processor in equidistant steps of $10\%$ in the interval $[0{,}100]\%$ to show the entire functionality and accuracy of our aspect-based calculation method. We exemplarily present the traces of the processor-specific utilization levels (by the *Intel Power Thermal Utility*) that we gained in the *SPECpower* benchmark $(SP1.2.8)$, as shown in Figure 125 and Figure 126, consider the same intervals marked by the vertical solid lines, and present the horizontal axes as the target throughput in Figure 129. In the upper graph, we present the actual utilization level in $[\%]$ calculated as a mean value of their particular interval that we simplify as a dashed-dotted red line by their corresponding label. In our example, the *SPECpower* benchmark specifies the target throughputs, which result in a maximal utilization level of $100\%$ and reduce them systematically. In the middle of the graph, we present the measured power consumptions $[W]$, simplified as mean values (traced by the *Intel Power Gadget*), and illustrated as a dashed-dotted red line by their corresponding labels. We simulate the processor power consumption on the basis of the measurement traces of the real utilization levels, but present our simulation results as mean values by the dashed-dotted red line and their corresponding labels. In our example, both (the measured and simulated) power consumptions are very close to each other, which results in nearly identical values, as listed in Table 73 and Table 74 in a more legible format. We observe a higher slope of the measured power consumption in Table 73 in comparison to the slope in Table 74 in which the *absolute differences* are at their highest value at the utilization levels between $40\%$ and $70\%$ in comparison to our simulation-based results. We further evaluate the processor-specific power development in Section 7.3 and check whether we require a linear or non-linear power method, considering various processors with different thermal design powers, additional p-states, or frequencies.

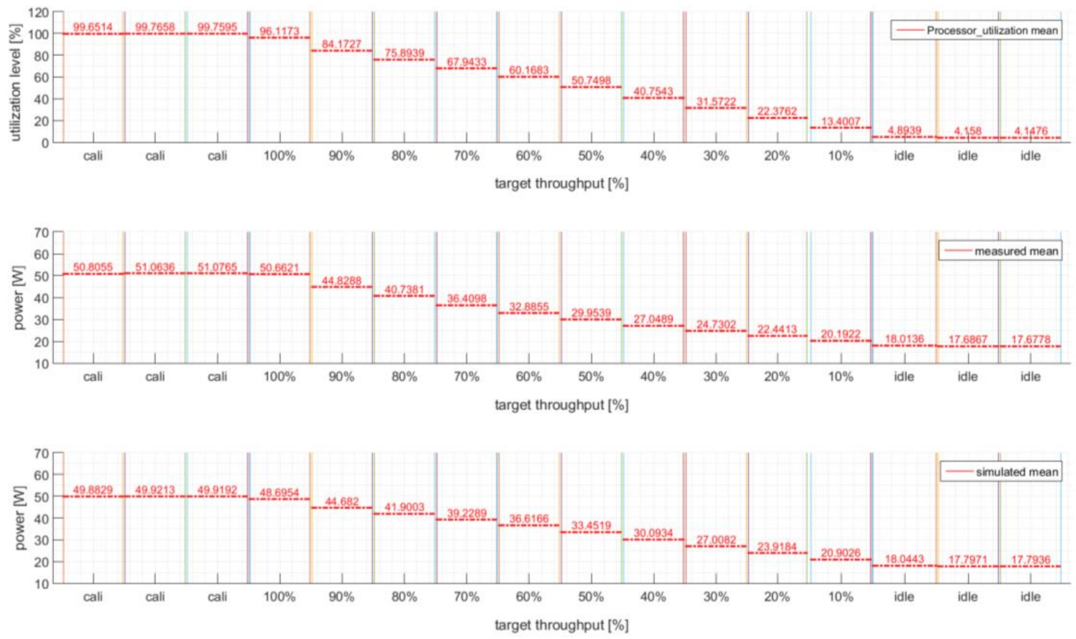


Figure 129: Processor power accuracy $(SP1.2.8)$

Table 73: Mean utilization level [%] and mean power consumption (measured vs. simulated) at target throughput $(calibration, 100\% - 60\%)$ in $(SP1.2.8)$

| | Target throughput [%] | | | | | |
|---|---|---|---|---|---|---|
| Mean Utilization levels [%] | Calibration | 100 | 90 | 80 | 70 | 60 |
| | 99.73 | 96.12 | 84.17 | 75.89 | 67.94 | 60.17 |
| Measured power consumption [W] | 50.92 | 50.66 | 44.83 | 40.74 | 36.41 | 32.89 |
| Simulated power consumption [W] | 49.91 | 48.70 | 44.68 | 41.90 | 39.23 | 36.62 |
| *Absolute difference* [W] | -1.01 | -1.96 | -0.15 | +1.16 | +2.82 | +3.73 |

Table 74: Mean utilization level [%] and mean power consumption (measured vs. simulated) at target throughput $(50\% - 10\%, idle)$ in $(SP1.2.8)$

| | Target throughput [%] | | | | | |
|---|---|---|---|---|---|---|
| **Mean** | **50** | **40** | **30** | **20** | **10** | **Idle** |
| **Utilization levels [%]** | **50.75** | **40.75** | **31.57** | **22.38** | **13.40** | **4.4** |
| **Measured power consumption [W]** | 29.95 | 27.05 | 24.73 | 22.44 | 20.19 | 17.79 |
| **Simulated power consumption [W]** | 33.45 | 30.09 | 27.01 | 23.92 | 20.90 | 17.88 |
| ***Absolute difference* [W]** | +3.50 | +3.04 | +2.28 | +1.48 | +0.71 | +0.09 |

Additionally, we analyze the *absolute difference* (in the mean) between the measurements and our simulation-based power, which rely upon the high-resolution of the utilization levels to consider all values of the specific times. We simulate the power consumption by a mean inaccuracy of approximately $+0.84W$, which is an $+3.6\%$ inaccuracy in relation to our measurements of the entire benchmark. In this example, we observe a median at $+0.06W$, a standard deviation by $+3.3W$, and a variance of $+10.9W$. Figure 130 presents the processor power consumption by the related normalized probability function (histogram) on the basis of the *absolute differences* in Watt.

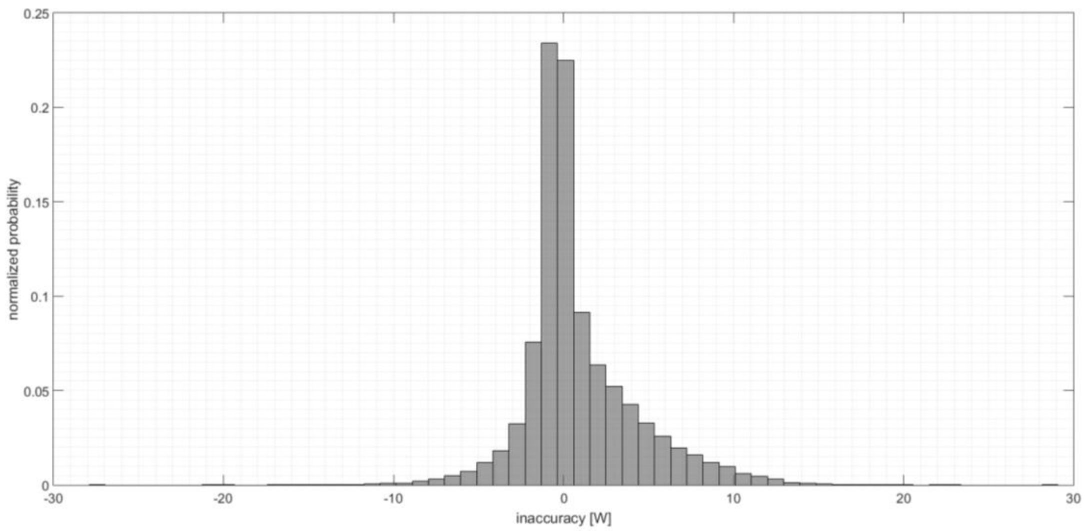


Figure 130: Processor power accuracy – a histogram (normalized probability) $(SP1.2.8)$

Table 75 lists the inaccuracy of our processor power model under various benchmarks concerning their *absolute* as well as *relative differences* between the simulated and measured values, both stated as mean[384] values. Our aim is an error rate less than ten percent, which we specify as precise enough, see Chapter 4. Generally, we can argue that our processor power model is sufficiently accurate when we neglect the explicit workload scenario, and for

[384] Mean values: $\mu$, average of all data values concerning the high-resolution of the utilization levels

simplicity purposes neglect the specific processor operations considering a few exceptions regarding the accuracy, e.g. $(SP2.3.36)$, $(MT16.z)$, and $(PM2.1.4)$. The mean utilization levels in the *SPECpower* benchmark $(SP2.3.36)$ do not behave the same way as we trace the remaining $(SPx.y.z)$ benchmarks. Herein, the utilization levels continuously toggle towards the maximal utilization level of $100\%$ at each target throughput. Our simulation model is insufficiently precise under these circumstances, as listed in Table 75 with a *relative difference* by $+32.24\%$. Additionally, we observe a *relative difference* of $+14.42\%$ at the *MemTest86* benchmark $(MT16.z)$, which primarily results from the power inaccuracy of the $(MT16.32)$ test case[385]. The benchmark does not fully utilize all the 16 processor-specific hardware threads after allocating approximately half of the memory capacity $(16GB)$ at $t = 9.6 * 10^4 s$ up to the entire memory capacity $(32GB)$ used at $t = 4.07 * 10^5 s$, which we exemplarily present in Figure 131 indicated by the memory utilization levels. We assume that the increasing memory capacity (addressing toward the end of the memory address range) and accessing (loading) a series of data onto the cache lines require more communication and consequently reduce the utilization level of the processor and the related power consumption. The remaining results of the *SPECpower* and *MemTest86* benchmark are sufficiently precise because their *relative differences* are less than $\pm 6\%$.

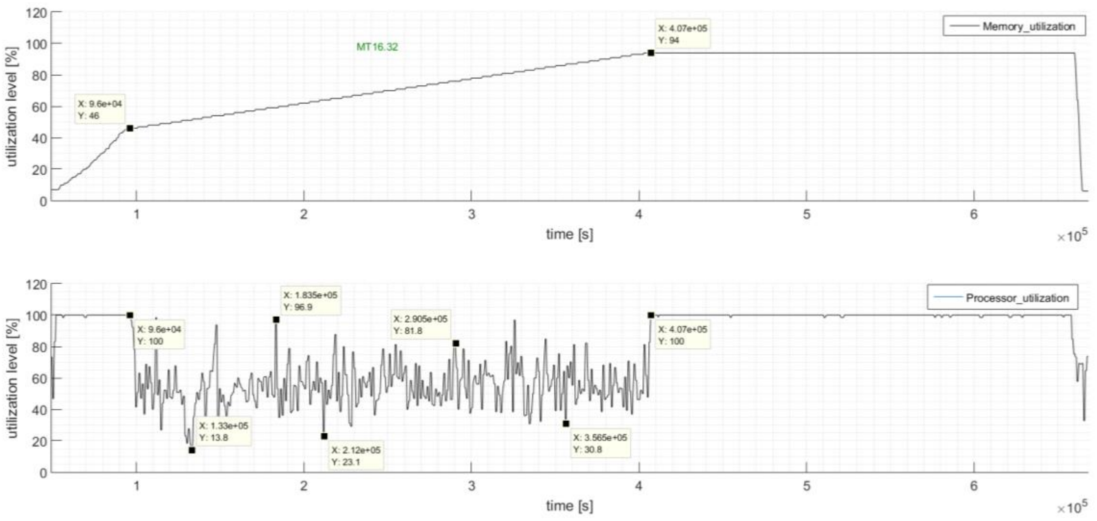


**Figure 131: Memory and processor utilization level $(MT16.32)$ – an excerpt of $(MT16.z)$**

We found that our *PassMark CPU* results are especially reliable and adequate because the *relative differences* are in the range of $[+1.5, +8.2]\%$, which is valid of the *PassMark Memory* benchmarks observing an accuracy between $+0.6\%$ and $+10.1\%$ when we neglect the results of $(PM2.1.4)$, see Table 75. In our $(PM2.1.4)$ benchmark, we expect the same tendency, such as in the remaining *PassMark Memory* benchmarks, but we observe extreme inaccuracies

[385] *MemTest86* test case: We execute the *MemTest86* in a virtual machine and adapt the memory capacity from $32GB$ to $1GB$.

because of the appearing middle-term (more than a few seconds) power peaks up to $190W$, which especially frequently occur at the end of the benchmark.

We demonstrate the precise simulation of the processor power considering the processor-specific utilization levels and processor-specific frequencies of the *Intel Xeon* architecture. We found that we could neglect the thread-specific data (utilization levels, frequencies) because the values are nearly identical and their deviations do not significantly affect the processor power consumption. The peak power consumption of a fully utilized processor depends upon the processor architecture and its integrated subsystems (see footnote 330) activated by the particular micro-benchmarks.

Table 75: Processor power accuracy – the simulated vs. the measured results

| | | Inaccuracy | |
|---|---|---|---|
| | | *Absolute difference* [W] (mean) | *Relative difference* [%] (mean) |
| ***SPECpower*** | $SP1.2.8$ | +0.84 | +3.56 |
| | $SP1.3.18$ | +1.32 | +5.28 |
| | $SP2.2.16$ | -4.54 | -1.90 |
| | $\boldsymbol{SP2.3.36}$ | **+6.78** | **+32.24** |
| ***MemTest86*** | $MT1.z$ | -0.01 | +0.41 |
| | $MT8.z$ | -0.80 | -1.39 |
| | $\boldsymbol{MT16.z}$ | **+4.73** | **+14.42** |
| ***PassMark CPU*** | $PC1.1.2$ | +0.71 | +3.43 |
| | $PC1.1.4$ | +0.02 | +1.50 |
| | $PC1.2.4$ | +0.29 | +2.51 |
| | $PC2.1.4$ | +1.15 | +5.65 |
| | $PC2.1.8$ | +0.62 | +2.56 |
| | $PC2.2.8$ | +1.23 | +8.24 |
| ***PassMark Memory*** | $PM1.1.2$ | +0.82 | +3.32 |
| | $PM1.1.4$ | +0.19 | +0.60 |
| | $PM1.2.4$ | +0.28 | +1.62 |
| | $\boldsymbol{PM2.1.4}$ | **+2.58** | **+24.02** |
| | $PM2.1.8$ | +0.04 | +1.74 |
| | $PM2.2.8$ | +1.52 | +10.13 |

***Processor Temperature Evaluation***

Moreover, we simulate the processor temperature on the basis of the processor power consumption (typically direct dependent upon the utilization levels). If a processor has a constant utilization level, the temperature will continuously increase due to the power dissipation, which is known as self-heating. The thermal development of the processor behaves similarly to the memory temperature behavior, and thus we specify various slopes of the increasing, decreasing, or steady utilization levels in our temperature-based methods considering the time delay because of the inertia. We neglect short-term peaks (less than one second) of the power consumption that do not influence the processor temperature and measure the processor-specific temperature[386] by the *Intel Power Gadget*.

Again, we analyze the *Intel Xeon E5-2650v2* processor (see Table 47) in our system under test while executing the *PassMark Memory* benchmark. The top graph in Figure 132 exemplarily shows the processor utilization levels (by the *Intel Power Thermal Utility*) and the middle of the figure presents the results of our processor temperature simulation (dash-dotted red line), which we evaluate according to our measurements (solid blue line) traced by the *Intel Power Gadget*. In our example, the *PassMark Memory* benchmark $(PM2.1.4)$ utilizes the processor below $20\%$ (in the mean $5\%$) until $t = 1.557 * 10^5 s$ and fully utilizes the processor at $T = [1.557 * 10^5, 1.784 * 10^5]s$, which we similarly observe in the remaining $(PMx.y.z)$ benchmark runs. The simulated temperature increases from $37.05°C$ up to $42.76°C$ at the time $T = [2.027 * 10^4, 1.144 * 10^5]s$, marginally decreases afterwards around $1°C$, but significantly increases to $50.95°C$ when the processor is fully utilized at the time $T = [1.558 * 10^5, 1.678 * 10^5]s$, as shown in Figure 132. Herein, we overestimate the temperature by a mean inaccuracy of approximately $+3.9°C$. This is an $+10.44\%$ inaccuracy in relation to our measurements, which is less precise than the rest of the *PassMark Memory* benchmarks.

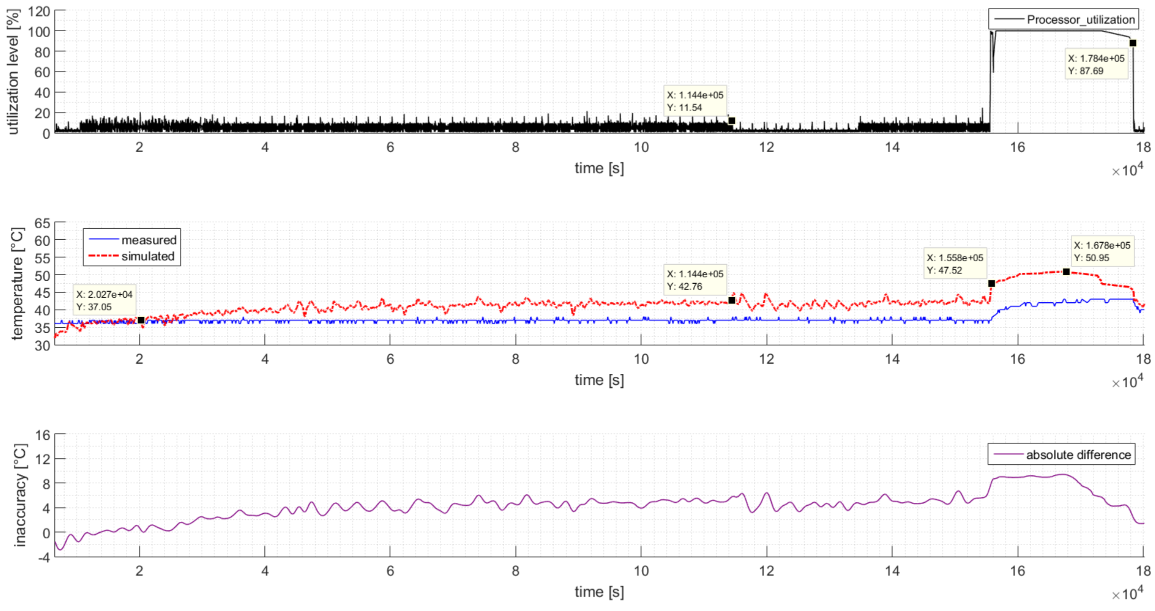


**Figure 132: Processor temperature accuracy $(PM2.1.4)$**

[386] Processor-specific temperature: We consider the same level of details of the processor temperature concerning the processor-specific utilization levels; otherwise, we have to trace the thread-specific data.

In contrast, the behavior of the *PassMark CPU* benchmark $(PC2.1.4)$ is different from the *PassMark Memory* benchmark, which fully utilizes the processor most of the time and may lead to a different thermal development of the processor. Therefore, we check whether our processor temperature model works sufficiently precisely concerning the influence of the changed utilization levels. The top graph in Figure 133 exemplarily shows the processor utilization levels (by the *Intel Power Thermal Utility*) of the identical server configuration used in $(PM2.1.4)$, but we execute the *PassMark CPU* benchmark $(PC2.1.4)$ instead. Figure 133 exemplarily presents the measured and simulated temperatures in the middle graph. Herein, we observe that the benchmark fully utilizes the processor nearly the entire time at $T = [1.552 * 10^4, 1.883 * 10^5]s$, however being interrupted by changing the micro-benchmarks. We observe that the processor temperature increases in the range of $[40.8, 49.0]°C$ when we fully utilize the processor, but measure a temperature decrease after executing the *floating-point* operations from $45°C$ down to $43°C$ at $T = [5.609 * 10^4, 5.915 * 10^4]s$. The lower graph of the figure presents the *absolute difference* in the mean (marked in purple) between the measurements and our simulation, in which we can observe an increasing temperature inaccuracy when executing the micro-benchmarks *prime numbers* or *extended instructions*. In our temperature model, we neglect the specific processor operations and simulate the thermal development on the basis of the utilization level. Nevertheless, we observe an *absolute* temperature difference (in the mean) by approximately $+1.67°C$ and a *relative difference* by around $+3.89\%$ inaccuracy in relation to our measurements of the entire benchmark.

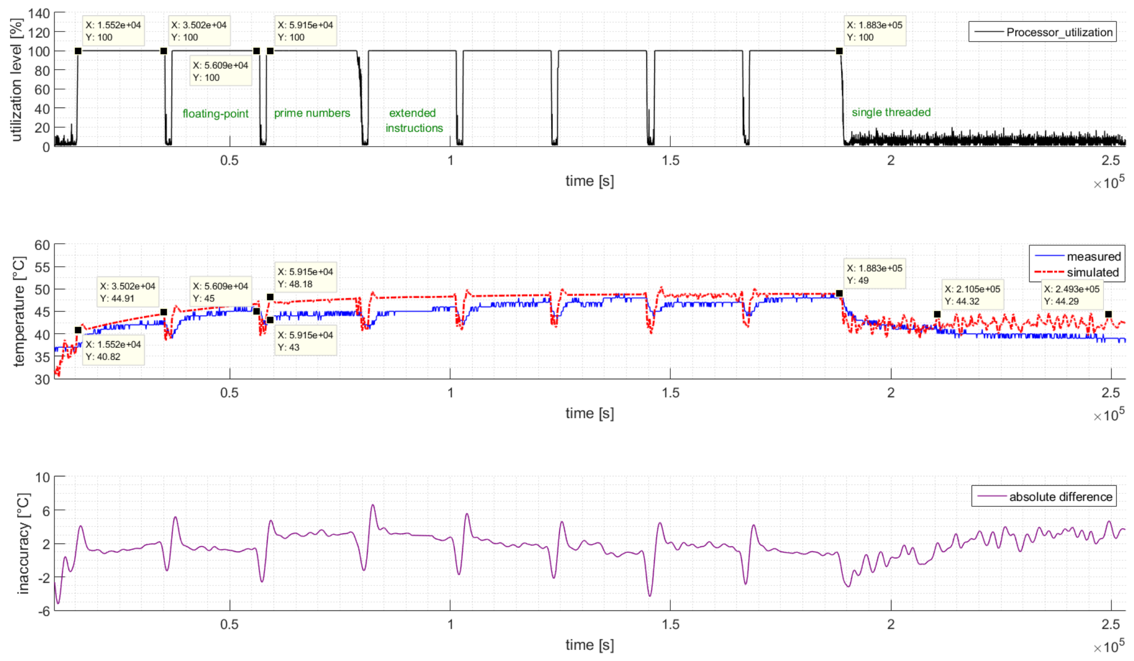


**Figure 133: Processor temperature accuracy $(PC2.1.4)$**

In the next section, we analyze our simulation-based results on the basis of more flexible utilization levels in comparison to the *PassMark* benchmarks. We exemplarily present the traces of the processor-specific utilization levels (by the *Intel Power Thermal Utility*) that we gained while executing the *SPECpower* benchmark ($SP1.2.8$). We exemplarily present our measurement results as mean values that are as simple to read as the results of Figure 132 or Figure 133. The horizontal axes in the three diagrams of Figure 134 illustrate the target throughput in $[\%]$ (marked by the vertical solid lines), beginning with the three calibration phases, followed by a sequence of target throughputs $100\%$ down to $0\%$ and ending by three idle intervals. We present the actual utilization levels $[\%]$ in the upper graph of Figure 134, calculated as a mean value of their particular interval (by a dashed-dotted red line and their corresponding labels). The middle graph of Figure 134 shows the measured temperatures $[°C]$, simplified as mean values (traced by the *Intel Power Gadget*), and illustrated as a dashed-dotted red line by their corresponding labels. We illustrate our simulation-based results $[°C]$ as mean values by the dashed-dotted red line and their matching labels in the lower graph of Figure 134. In our exemplary analysis, both (the measured and simulated) temperatures are very close to each other, but in particular vary around $4.61°C$ in the idle case. In our processor temperature model, we take the ambient temperature into account, which may result in a higher temperature. Furthermore, we neglect the fan-based cooling effect on the processor in this section because we consider the processor as a single separate component decoupled from the entire system and management strategies.

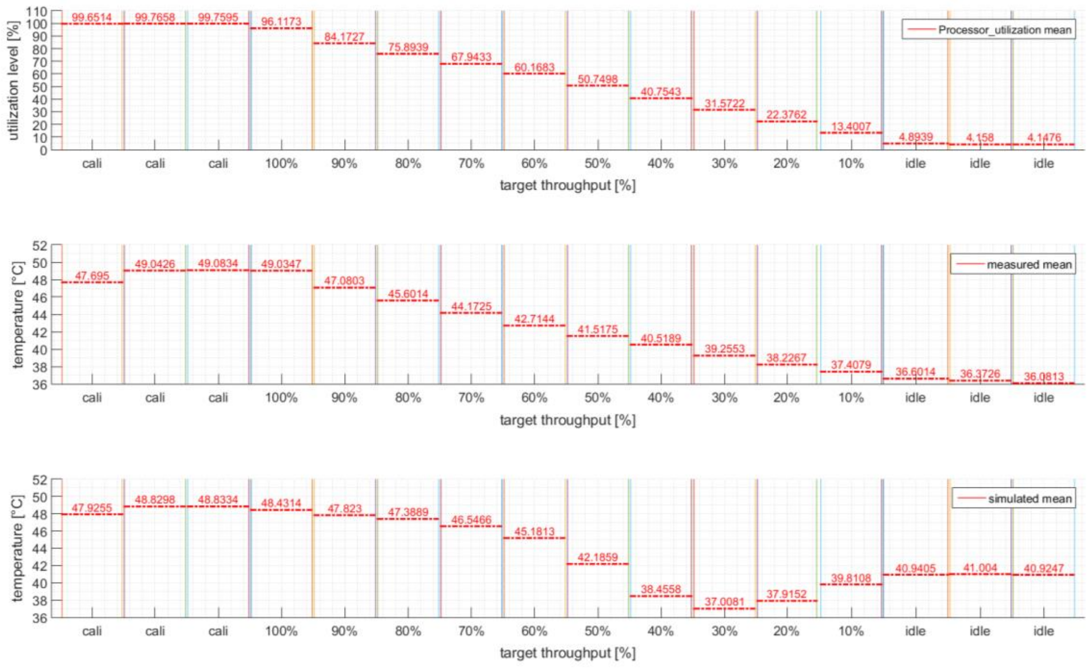


Figure 134: Processor temperature accuracy ($SP1.2.8$)

**Table 76: Mean utilization level [%] and mean temperature (measured vs. simulated) at target throughput $(calibration, 100\% - 60\%)$ in $(SP1.2.8)$**

| | Target throughput [%] | | | | | |
|---|---|---|---|---|---|---|
| **Mean** | **Calibration** | **100** | **90** | **80** | **70** | **60** |
| **Utilization levels [%]** | **99.73** | **96.12** | **84.17** | **75.89** | **67.94** | **60.17** |
| **Measured temperature [°C]** | 48.61 | 49.03 | 47.08 | 45.60 | 44.17 | 42.71 |
| **Simulated temperature [°C]** | 48.53 | 48.43 | 47.82 | 47.39 | 46.55 | 45.18 |
| ***Absolute difference* [°C]** | -0.08 | -0.60 | +0.74 | +1.79 | +2.38 | +2.47 |

**Table 77: Mean utilization level [%] and mean temperature (measured vs. simulated) at target throughput $(50\% - 10\%, idle)$ in $(SP1.2.8)$**

| | Target throughput [%] | | | | | |
|---|---|---|---|---|---|---|
| **Mean** | **50** | **40** | **30** | **20** | **10** | **Idle** |
| **Utilization levels [%]** | **50.75** | **40.75** | **31.57** | **22.38** | **13.40** | **4.4** |
| **Measured temperature [°C]** | 41.52 | 40.52 | 39.26 | 38.23 | 37.41 | 36.35 |
| **Simulated temperature [°C]** | 42.19 | 38.46 | 37.01 | 37.92 | 39.81 | 40.96 |
| ***Absolute difference* [°C]** | +0.67 | -2.06 | -2.25 | -0.31 | +2.40 | +4.61 |

In our example, we observe an *absolute* temperature difference (in the mean) by approximately $+1.19°C$ and a *relative difference* by around $+3.17\%$ inaccuracy in relation to our measurements of the entire benchmark. Figure 135 presents the processor temperature accuracy by the related normalized probability function (histogram) on the basis of the *absolute differences* in degree Celsius. We overestimate the processor temperature and observe a median at $+0.89°C$, a standard deviation by $+2.42°C$, and a variance of $+5.87°C$.

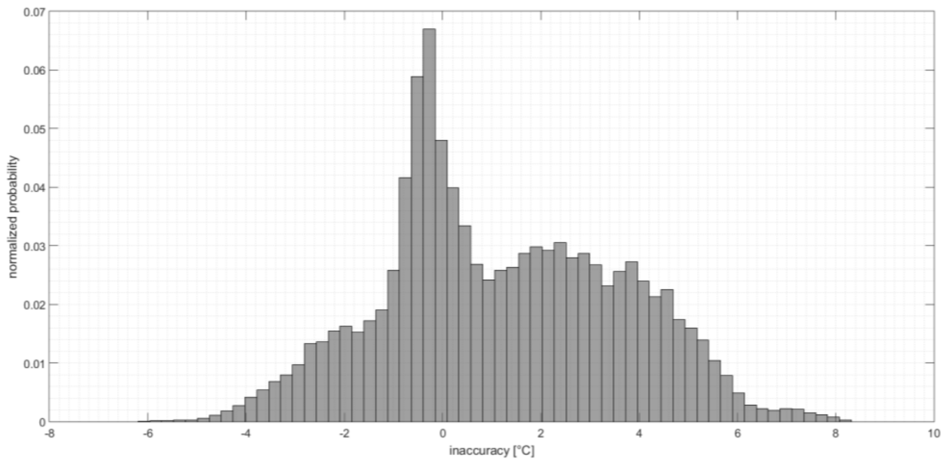


**Figure 135: Processor temperature accuracy – a histogram (normalized probability) $(SP1.2.8)$**

Table 78 lists the inaccuracy of our processor temperature model under various benchmarks concerning their *absolute* as well as *relative differences* between the simulated and measured values, both stated as mean values. Our aim is an error rate less than ten percent, which we specify as exact enough, see Chapter 4. Generally, we can argue that the results are sufficiently precise with a few exceptions, such as in $(SP2.3.36)$ and $(PM2.1.4)$. We observe in the *SPECpower* benchmark $(SP2.3.36)$ that the utilization levels do not behave the same way as we trace in the leftover benchmarks $(SPx.y.z)$, because the utilization levels continuously toggle towards the maximal utilization level of $100\%$ at each target throughput, which especially result in inadequate power consumptions and corresponding temperatures by a *relative difference* of $+11.02\%$. The remaining results of the *SPECpower* and *MemTest86* benchmarks are sufficiently accurate because their *relative differences* are less than $\pm 9\%$. We found that our *PassMark CPU* and *PassMark Memory* results are especially reliable and adequate because the *relative differences* are in the range of $[-1.81, +3.89]\%$ and $[-1.77, +10.44]\%$, although the $(PM2.1.4)$ power consumption is inaccurate.

Table 78: Processor temperature accuracy in comparison to the measurements

| | | Inaccuracy | |
|---|---|---|---|
| | | *Absolute difference* [°C] (mean) | *Relative difference* [%] (mean) |
| ***SPECpower*** | $SP1.2.8$ | +1.19 | +3.17 |
| | $SP1.3.18$ | +2.39 | +5.85 |
| | $SP2.2.16$ | +3.4 | +8.56 |
| | $SP2.3.36$ | **+4.19** | **+11.02** |
| ***MemTest86*** | $MT1.z$ | +2.42 | +6.62 |
| | $MT8.z$ | +3.58 | +8.84 |
| | $MT16.z$ | +3.07 | +7.51 |
| ***PassMark CPU*** | $PC1.1.2$ | -0.51 | -1.21 |
| | $PC1.1.4$ | -0.76 | -1.81 |
| | $PC1.2.4$ | -0.50 | -1.15 |
| | $PC2.1.4$ | +1.67 | +3.89 |
| | $PC2.1.8$ | +1.34 | +3.08 |
| | $PC2.2.8$ | +1.48 | +3.40 |
| ***PassMark Memory*** | $PM1.1.2$ | +0.27 | +0.58 |
| | $PM1.1.4$ | -0.61 | -1.77 |
| | $PM1.2.4$ | -0.01 | -0.15 |
| | $PM2.1.4$ | **+3.90** | **+10.44** |
| | $PM2.1.8$ | +3.02 | +7.95 |
| | $PM2.2.8$ | +3.36 | +8.90 |

### 7.2.5 Conclusion

The presented results of the previous sections show that our simple aspect-based component models can be used to predict in sufficient accuracy the power and temperature in a wide range of flexible heterogeneous workload scenarios considering category-specific utilization levels. We identify that the memory-specific read-to-write ratio has a significant impact on the power simulation, which can reduce the over-estimation of the memory power consumption. We observe the highest memory power at the processor-intensive workloads, e.g. *PassMark CPU,* in comparison to the memory-intensive workloads, such as the *PassMark Memory* benchmark, because of the communication between the processor and memory. We define the memory model as a state-based approach, like the respective authors specified in [RL 2007, Riv 2008, BC 2010], but we particularly consider the memory's interaction by the read-to-write ratio in the certain workload scenario. The authors in [MAC et al. 2011] predict the memory power by a mean relative error of $[+4,+8]\%$ across various micro-benchmarks, including a single core processor. Our simulation results are equivalent to the results presented in [KCB et al. 2013], whereby the accuracy of the memory current and obtained power consumptions are in a range of $[+2.1,+14.02]\%$. We get comparable results to those achieved by the authors of [KCB et al. 2013], but involve less effort because we group the $IDD$ values instead of considering every $IDD$ state that are defined in the spreadsheets. The precise power consumption builds the base of the temperature simulation, which we differentiate into a steady state, an increasing, and a decreasing temperature development. We neglect short-term peaks (less than one second) of the power consumption, which have no impact on the memory and processor temperature because of the thermal inertia. The authors of [APL et al. 2008] simulate the memory temperature with an accuracy of less than $+5\%$ considering various micro-benchmarks and performance counters of the hardware-specific events. Our simulation-based thermal results are adequately precise and require less effort in analyzing the memory modules. Moreover, we identify that the thread-specific utilization levels and frequencies of the processor (*Intel Xeon* architecture[387]) do not improve the accuracy of the processor power consumption, but we demonstrate sufficiently precise results of our decoupled component models, which are as accurate as the approach in [MAC et al. 2011] of the multi-core processor. The authors of [MAC et al. 2011] estimate the multi-core processor power by an average accuracy between $[+10,+14]\%$, which is refined in case of a single core processor power by an accuracy of $[+2,+6]\%$. In the work of [Riv 2008], the authors, especially analyzed the *Intel Xeon* architecture and predict the power in their utilization-based models by a mean accuracy of less than $\leq 15\%$ considering various processor-intensive micro-benchmarks. Their performance-counter-based models are more accurate than the utilization-based models as identifiable by a mean accuracy of approximately $+10\%$. We define the component-specific aspects as separate functions considering the entire utilization levels (processor-specific) instead of the instruction sets, particular operation traces, or activities on a cycle-by-cycle basis as done in [Che 2006, Riv 2008, RRK 2008]. We found that using the

[387] *Intel Xeon* architecture: *Intel Xeon* processor *E3/5/7*-xxxx *v1-5* family

integrated subsystems of the processor has a relevant impact on the processor power consumption, which is observable while executing the various micro-benchmarks of the *PassMark CPU* benchmark. Nevertheless, we simulate the processor power and temperature that are only based on the utilization levels and consider a mean usage of the subsystems in which we neglect the specific operations and the fan-based cooling effect, but also simulate equivalent results (sufficiently accurate precision and accuracy) concerning our measurements on the real server system. Our aspect-based component models require marginal component-based data, such as characterization, ambient temperature, and utilization levels. We achieve the same level of accuracy as the counter-based power models described in [Bel 2001] by around $+10\%$ using two event counters or $+5\%$, considering additional counters. We do not study the particular counters, which result in less effort in comparison to the counter-based models by [Han 2007]. Our accuracy is as good as the accuracy of [Han 2007, HS 2007] within the range of $[-4, +10]\%$. Their thermal model has an accuracy in the range of $[-1.3, +3.4]°C$ considering the *Intel Pentium III* processor, which is nearly identical with our temperature accuracy. We assume that creating rough models considering the technical specification can be as adequate as highly detailed models, which, in consequence, creates less computational effort. Generally, our accurate and precise power as well as temperature results leads to the assumption that we can avoid over-provisioning in the case of the PSU sizing.

## 7.3 Analysis the Impact on Changes of the Component Characteristics

### 7.3.1 Objective

Our aim is the verification and evaluation of the concepts' operating principles regarding the flexible changes of the component characteristics to support a wide range of various server systems. In this section, we primarily show the scalability of the category-based and aspect-based characterization considering the certain adjustments of the technical specification $CH_{TS}$, static characteristic $CH_{CFG}^{ST}$, or dynamic characteristics $CH_{CFG}^{DY}$. We alternate the component characteristics, analyze the accuracy, and check whether we simulate adequate results. We want to find the limits of our model because of the chosen relevant characteristics and check their universal applicability on future and uncertain systems. Additionally, our component-based models shall reduce the measurement effort of actual server systems.

### 7.3.2 Evaluation Criteria

Our models instantaneously have to react rapidly and efficiently to the changes of the characteristics. We assume that if our component models flexibly respond to the particular characteristic adjustments, the entire simulation model will provide adequately aspects-based results. In this analysis, we answer the following questions:

- How much can we trust our component-based models and their related aspect-based calculation methods when we change the internal component characteristics forced by a recent external server configuration?
- What are the consequences (impacts) of the characteristic changes in our component models concerning the accuracy and precision?

As a prerequisite, we have to answer the question whether the component-based models flexibly react upon varying characteristics. As described in the concept chapter, we empirically analyze the component-specific characteristics in academia and industrial practice to determine their significance on the certain aspect, contribute the most variability, and reduce the simulation output uncertainty, which is a further precondition. Section 7.2 indicates that our component models are working in an accurate manner because we already adjust some of the memory characteristics, such as the capacity. In this section, we concentrate upon the varying characteristics and study the particular components to demonstrate the functionality of our concept. We evaluate the accuracy of our methods by comparing our simulation-based results with the measurement values gained from the components in a real server system on the one hand, and with the vendor-specific spreadsheets on the other. Fujitsu started a couple of measurement processes over the years to trace especially the power consumption for a large number of various memory modules and *Intel* processors. We have confidence in the quality, comparability, and accuracy of the measurement procedure and therefore consider the Fujitsu-internal database results in our evaluation. Our evaluation criteria are the *absolute* and *relative differences* between the simulation-based values and either the measurement results gained by Fujitsu or the vendor-specific spreadsheets in terms of availability and a relatively high level of comparability (nearly identical characteristics). We expect that the purely spreadsheet-based approaches overestimate the current aspect more than our aspect-based calculation methods. On the other hand, when simulating the power consumption, we require an over-prediction less than $+10\%$ in relation to the real-life measurement. We concentrate on the power simulation in our analysis and partly consider the thermal results because of the limited available data. We study a sequence of diverse memory modules and processors.

### 7.3.3 Experimental Setup

We specify the general evaluation environment and measurement infrastructure in Sections 7.1.1, 7.1.2, and 7.1.3. Herein, we specify the experimental setup of the *impact on characteristic changes*, which is nearly the same with regard to the procedure in Section 7.2.3. We separately evaluate the category-specific components regarding their accuracy of the aspect-based calculation methods, as shown in Figure 105. Herein, we exclusively consider the specific component and call the particular *calc_category()* method, provided by the component model, which is isolated and encapsulated from the other components. This approach simplifies the evaluation and speeds up the simulation because we analyze each component and its characteristic changes separately. In general, we analyze the recent external server configuration when we start the entire simulation, but we adjust the characteristics of each component in an isolated manner in our experimental setup to speed up the evaluation. Additionally, we ignore the relations or communications between the various components to encapsulate the certain component from each other. In this evaluation section, we disable the *system-wide optimization engine* because we analyze the results of our component-based models concerning the flexible characterization. In contrast to Section 7.2, here we analyze various *Intel* processors as well as memory modules, whereby we consider various families, generations, or series as technical specification or the memory density, capacity, or die as different static characteristics that could originate from various manufacturers. Therefore, we adjust the component configuration as an input parameter of the simulation framework or isolated component model. In order to ensure the comparability of the simulation results, we modify the stimuli to produce a very similar situation considering the utilization levels and the component characterization. We calculate the *absolute* as well as *relative difference* regarding our simulation-based results.

### 7.3.4 Results and Analysis

We exemplarily present the results of our memory and processor models to evaluate the parameter sensitivity of our simulation model and present the simulation-based values resulting from the changing simulation parameters. Herein, we concentrate upon the power-specific calculation methods because the power consumptions of the various components are accessible everywhere and available to anyone. In contrast, the component-specific temperatures are not well analyzed regarding the variability of their characteristics. Therefore, we evaluate our thermal models that are based on the power consumption and the utilization levels, see Section 7.2.4. First, we present the adapted power consumptions of our simulation model considering various characteristics on condition that we only change the particular simulation parameter without modifying the models. We analyze their accuracy (evaluation criteria: the absolute and relative differences) with regard to the measurement results gained by Fujitsu. The experimental results of Fujitsu are stored in an internal database and not accessible to the public, which, on the other hand, are used to simulate the component-specific power in the commercial tools. Furthermore, we compare a subset of our simulation-based results in accordance with the academic approaches or vendor-specific spreadsheets that are suitable regarding a high level of comparability (nearly identical characteristics).

#### *Memory Power Evaluation*

At first, we concentrate upon the two different memory modules considering the configurations $(C70, C71)$ that we analyze in the test cases $(PC1.1.4)$ and $(PM1.1.2)$ in Section 7.2. In our memory power evaluation, we exemplarily analyze a selection of characteristics[388] that we vary as a subset of the possible parameters to show the sensitivity and accuracy of our simulation model. Table 79 lists the memory characteristics of the SUT

[388] Characteristics: Table 42 lists the detailed characteristics

modules, which we consider as the particular simulation parameters. The authors of [RLG et al. 2008] analyzed the varying designs of the $DDR2$ memory modules considering the idle power of diverse vendors. In this section, we concentrate upon the maximal power consumption of the $DDR3$ memory modules starting with the vendor-specific evaluation.

Table 79: Memory modules $(C70, C71)$ of the SUT – characteristics considered as simulation parameters

| Memory characteristics | Memory module $(C70)$ | Memory module $(C71)$ |
|---|---|---|
| Vendor | {'Micron'}; {'C'} | {'Qimonda'}; {'D'} |
| Capacity (size) [GB] | {'4GB'} | {'2GB'} |
| Density [GB] | {'4GB'} | {'2GB'} |
| Die (component revision) | {'D'} | {'D'} |
| Fabrication size [nm] | {'44nm'} | {'56nm'} |
| Synchronization mode | {'registered'} | {'registered'} |
| Module ranks, rank linking (data width) | {'1R'}; {'SR'}, {'x4'} | {'1R'}; {'SR'}, {'x4'} |
| Timings | {'11'} | {'7'} |
| Error correction | {'ECC'} | {'ECC'} |
| Frequency [MHz] | {'800'} | {'533'} |
| Voltage [VDC][389] | {'LV'}; 1.35VDC | {'STD'}; 1.5VDC |
| Transfer rate / throughput [MHz] | {'1600'}; {'PC3-12800'} | {'1066'}, {'PC3-8500R'} |

Instead of considering all variations in each $IDD$ state, we group the major $IDD$ states into the idle state and the active state, which we further distinguish in the refresh mode or read-to-write mode. In [KCB et al. 2013], the authors analyze every current variation in a single memory module, which we ignore as the impact of variations in the power consumption and are negligible in the variation of clusters.

We separately evaluate both memory modules $(C70, C71)$ because their characteristics differ from each other significantly, such as the capacity, density, or frequency. For the purpose of comparability, we temporary split the two memory modules into different groups, such as $(C70, C72, C73, C75)$ and $(C26, C71, C74, C76)$. Appendix A3e provides further details of the components and their characteristics. The eight bars to the left side of the Figure 136 present the measured and simulated results of the four memory modules $(C70, C72, C73, C75)$ with $4GB$ capacity that have almost identical characteristics, but are from several different vendors[390]. We fully utilize the memory modules by the synthetic *PassMark Memory* workload and use our default read-to-write ratio. We observe that the $4GB$ memory module of the

[389] VDC: volts direct current

[390] Different vendors: labeled in letters $A, B, C, D, E, F, G$ because we gained the particular power consumptions of the vendors that are confidential

vendors $D$ and $C$ consume nearly twice as much power as the memory modules of the vendors $A$ and $E$ in the measurements as well as in the simulation. The *absolute differences* in our simulated power in comparison to the measured values are in a range between $+0.03W$ and $+0.62W$ of the $4GB$ memory module. Figure 136 also presents the comparative values of the SUT-specific $2GB$ memory module $C71$ that are visualized in the six bars to the right side of the module, grouped as $(C26, C71, C74, C76)$. In contrast to the $4GB$ configurations, the power consumptions of the $2GB$ memory modules of the vendor $E$ and $C$ show an opposite trend, but our observation between the vendors $D$ and $A$ remain unchanged. The *absolute difference* of the $2GB$ memory modules between the measurements and our simulation is in the range of $[+0.03, +0.22]W$, which are sufficiently accurate results of our model. Herein, we implicitly show that our simulation model reacts on changing capacity and density. We observe a similar trend of the results by [RLG et al. 2008] in which the maximal power consumption depends upon the vendor and varies in a wide range. Some differences of the power consumption may occur, because we cannot consider the capacity as a separate characteristic. In our preceding analysis, we found that the memory capacity relies upon the rank, rank linking, and density. In order to reduce the mutual influence, we consider the particular modules, as shown in Figure 136. In addition, the authors of [KCB et al. 2013] state that the power consumption varies approximately $20\%$ through the various vendors, which is observable in Figure 136. In our evaluation, we concentrate upon the $DDR3-SDRAM$[391] memory and consider the dependencies of the various sizes and ranks on the power consumption that the authors of [XTB 2007] analyzed of the SRAM-based[392] memory modules.

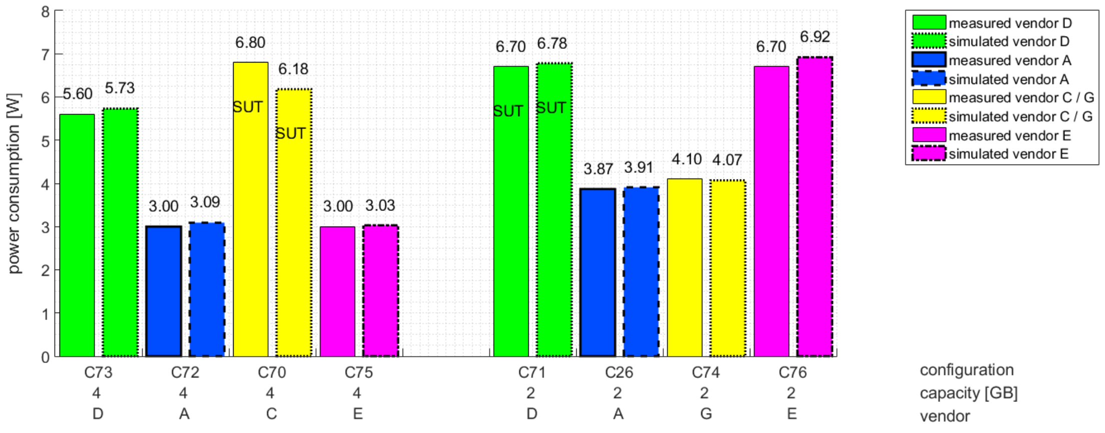


**Figure 136: Vendor-specific power consumption of the memory modules $(C26, C70 - C76)$**

In the next section, we change the memory frequencies, which we define as the dynamic characteristic and evaluate the impact on a single change in the power consumption. We do not have adequate measurable results concerning varying frequencies of the SUT memory

[391] DDR3-SDRAM: synchronous dynamic random-access memory
[392] SRAM: static random-access memory

modules and therefore we choose the $2GB$ memory modules $C26$ and $C24$ of vendor $A$ to show the relevance and effects on the power consumption. Again, we fully utilize the memory modules by the synthetic *PassMark Memory* workload while varying the memory frequencies. In our simulation model, we consider the baseline frequency of $533MHz$ for the memory modules $C26$ and $C24$. We continuously increase the frequency following the predefined steps of $[533,667,800,933]MHz$. The eight bars to the left side of Figure 137 show the $C26$ module; the eight bars to the right side present the $C24$ module as measured as well as simulated power values. We observe that a higher memory frequency results in an increasing power consumption. In our example, we simulate the power consumptions, which are higher than the measured values, but sufficiently accurate because the *absolute differences* are in a range of $[+0.02,+0.25]W$.

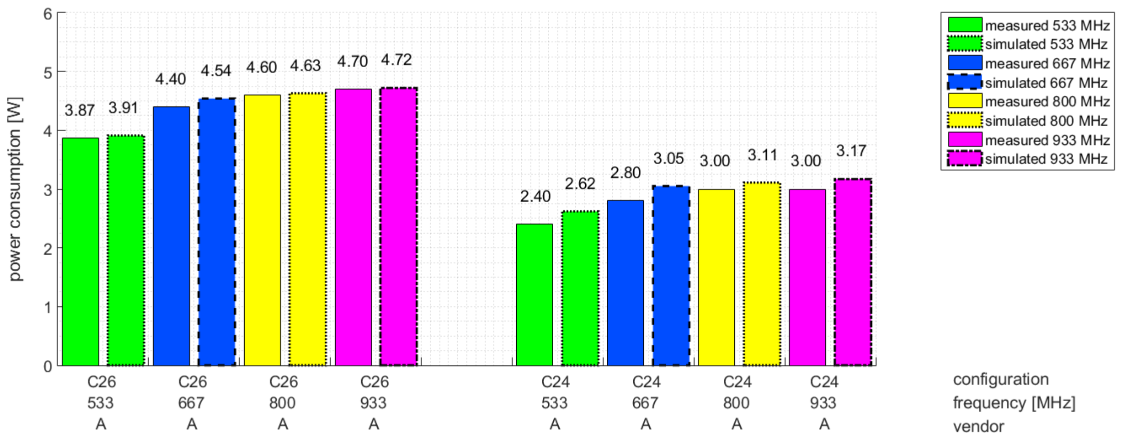


**Figure 137: Frequency-specific power consumption of the memory modules $(C26, C24)$**

Besides the capacity, density, vendor, or frequency, we found that the die types are also significant regarding the memory power consumption. We study the power consumption of the various die types in order to understand the future trends of memory power consumption. We assume a continuous development of memory modules.

Figure 138 exemplarily shows the three memory modules $C14$, $C20$, and $C26$ that are almost identical, but differ with their die types $(B, C, D)$. We simulate the three memory modules considering the *PassMark Memory* workload and fully utilize them to analyze the dependency on the die. In principle, we can simulate the same module and change their die type, but we gained the explicit measurement results of these three individual memory modules. We observe that the $D$-Die memory module $(C26)$ consumes less power than the $C$-Die $(C20)$ or $B$-Die $(C14)$. Our simulation-based results are less exact when we only consider the changing die types as a single parameter. In this example, we observe an *absolute difference* between $-0.7W$ and $+0.04W$ that is more accurate concerning the over-estimation, but is less precise in the under-estimation in comparison to the frequency-based results, for instance. Nevertheless, we can predict future components in our simulation model that follow the same trend as the memory modules before. The authors of [Vog 2010] present the scaling progresses through the memory generations, such as fabrication size, capacity, generation, or family regarding the power, voltage, timing, and energy trends that we consider in our approach.

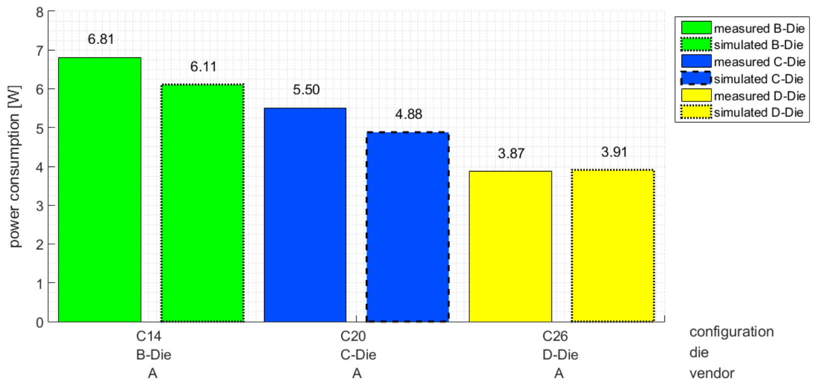


**Figure 138: Die-specific power consumption of the memory modules $(C14, C20, C26)$**

We exemplarily show that our simulation model reacts on the characteristics of the memory modules that influence the power consumption. In the majority of the cases, the characteristics influence each other and cannot be simulated separately. We evaluate the memory modules $(C77, C78)$ of the academic approaches [HCE et al. 2011] and [LEU et al. 2010] to show the applicability and reliability of our simulation-based approach. Accordingly, we specify the memory modules in our simulation model on the basis of the descriptions of the scientific papers, but we guess several parameters as default assumptions that are not documented in detail. In Figure 139, we observe that our simulation-based results are higher than the results gained from the academic approaches and lower than the listed power consumption of the spreadsheets[393]. The vendors provide the worst-case values (e.g. power, temperature) in their spreadsheets, which is a common industrial practice [Fuj 2014] and generally leads to an over-estimation. The *absolute differences* of our simulated power consumption in comparison to the academic approaches are between $+0.58W$ and $+0.73W$, which may occur because of the missing memory-specific details concerning a specific workload, such as the execution of various instructions or patterns. We found that our simulation model is sufficiently precise in relation to the detailed cycle-by-cycle or instruction-based models.

[393] Spreadsheets: the commercial tools usually consider the spreadsheet values

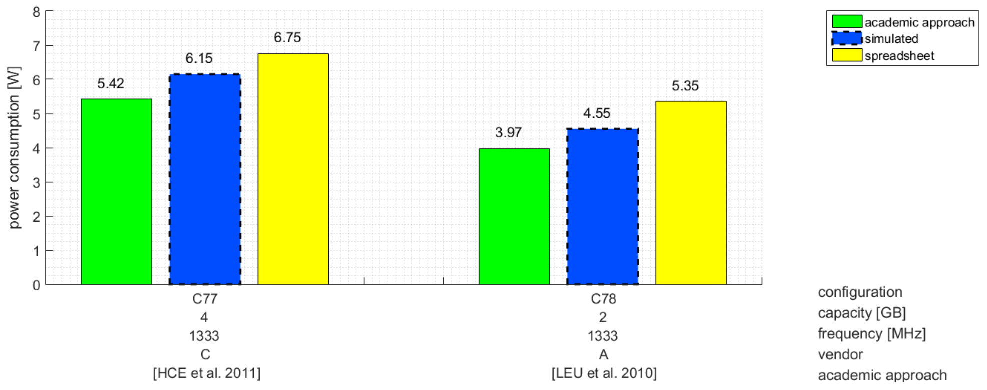


**Figure 139: Comparison of the memory power simulation $(C77, C78)$ concerning the academic approaches [HCE et al. 2011, LEU et al. 2010]**

### *Processor Power Evaluation*

In this section, we evaluate different processor configurations, considering various processor characteristics, as a subset of possible simulation parameters to show the sensitivity of our simulation model, see Table 44. We describe our results of the *Intel Xeon* architecture - the third generation, code name *Ivy Bridge*. In our analysis, we exemplarily simulate the *E5-2600* product family of the *E5-v2* processor family starting with the *E5-2650v2*. At first, we analyze the power gap between the spreadsheet-based estimation, the commercial tools, the measurements, and our simulation results. Afterwards, we evaluate the *absolute* and *relative difference* (in mean) between the measured and simulated values to show the functionality and flexibility of our approach. Our challenge is to improve the processor power calculation, especially at lower utilization levels, which is relevant for the memory-bounded workloads typical for a data center. Furthermore, we assume a continuous development of the processor family and series to define the power consumption in an early design phase more precisely.

In contrast to Section 7.2.4, we exemplarily present a detailed power analysis considering the exact characteristics that are relevant to simulate the entire power consumption of the processor and concentrate upon the processor-specific p-states that correspond to the processor-specific utilization levels. Table 80 lists the processor characteristics that we consider as the particular simulation parameters of *Intel Xeon E5-2650v2* configuration $(C18)$.

**Table 80: Processor $(C18)$ of the SUT (*Intel Xeon E5-2650v2*) – characteristics considered as simulation parameters**

| Processor characteristics | Processor $(C18)$ |
|---|---|
| L3 cache [MB] | `{'20MB'}` |
| L2 cache [KB] | `{'2048KB'}` |
| Semiconductor technology (TDP) [W] | `{'70W'}` |
| Vendor | `{'Intel'}` |
| Architecture | Intel *XEON E5* |
| Generation | Ivy Bridge EP |
| Family | *E5-2600v2* |
| Series | `{'E5-2650v2'}` |
| Cores / active cores (hyper-threading) | `{'8C'}, {'16T'}` |
| Frequency [GHz] | `{'2.1'}`, turbo `{'2.3'}` |
| Transfer rate [GT/s, MHz] | `{'8.0GT/s'}, {'1600MHz'}` |

In our analysis, we consider synthetic utilization levels of the processor in equidistant steps of $10\%$ in the interval $[0,100]\%$ as an input parameter of our simulation. At the same time, we assume an unchanged ambient temperature. This might negatively influence the accuracy of our processor power simulation. However, we make this assumption for two reasons. First, the ambient temperature is not under our control; second, we want to keep our model simple. As described in the concept chapter, we simulate the processor power in a manageable power interval considering the thermal design power (TDP) and the minimal power in relation to the amount of p-states as well as the processor family (series) as static characteristics. Figure 140 exemplarily presents the simulation-based results of the processor power (dash-dotted red line) in comparison to the measurement trace of the *Intel Power Thermal Utility* (solid blue line)[394], the data gained from the commercial tools[395] (dotted magenta line), or vendor-based data determined in the spreadsheets (dashed black line). We simulate the power consumption of the *Intel Xeon E5-2650v2* $(C18)$ as a linear function[396] of the utilization levels, which is nearly identical to the measurements. In contrast to our studies at the *SPECpower* benchmarks, we observe a higher *absolute difference* between the simulated and the measured values by a mean inaccuracy of approximately $+1.39W$, which is a $+6.72\%$ *relative difference* (in mean). The *Intel Xeon E5-2600* specification defines a nearly constant processor power in the range of $[58{,}70]W$. In addition, we observe significant differences between the

[394] Comparison to the measurement trace: detailed accuracy of $(C18)$, such as the frequency analysis, is presented in Section 7.2.4

[395] Commercial tools: we found equivalent configurations of the processors $(C1, C3, C7, C18)$ in the *Dell Energy Smart Solution Advisor (ESSA), http://www.dell.com/calc*

[396] Linear function: the processor power is non-linear for processors with more than approximately 15 p-states

realistic measured/simulated power values in comparison to the commercial tools, which overestimate the power especially at the full utilization levels. In our example, the measured/simulated power consumption at the full utilization level is approximately $50.59W$ but nearly $74W$ calculated by a commercial tool and approximately $70W$ defined by the spreadsheet. We observe that the accuracy of the power consumptions gained from the commercial tools is more precise when the utilization levels decrease. Thus, the *absolute difference* between the measured values and the commercial tool is $+15.26W$ at $50\%$ utilization level. At the same time, the simulated values have the highest level of inaccuracy of approximately $+3.46W$. We observe that the minimal power is nearly identical at all curves besides the spreadsheet-based power. Moreover, Appendix A3h provides the detailed power comparisons of the processors $(C1)$, $(C3)$, and $(C7)$, but we generally observe that the processor power consumptions of the spreadsheets are inadequate concerning their accuracy in comparison to the measurements, as analyzed in [DSC 2006, New 2008, Fuj 2014]. The results demonstrate the necessity of more precise estimation of the processor power consumption in industrial practice and show the improvements when we simulate the processor power consumption. We can easily reduce the power gap between the spreadsheet-based method and the measurements by using our simulation model to keep the inaccuracy as low as possible. We observe the best improvements of the spreadsheet-based approach at higher utilization levels, such as in a range of $[50 - 100]\%$. The minimal power of the processor is nearly identical with the commercial tools, simulations, or measurements. We can avoid over-provisioning because of our more precise power simulation in comparison to the commercial tools and spreadsheet-based approaches.

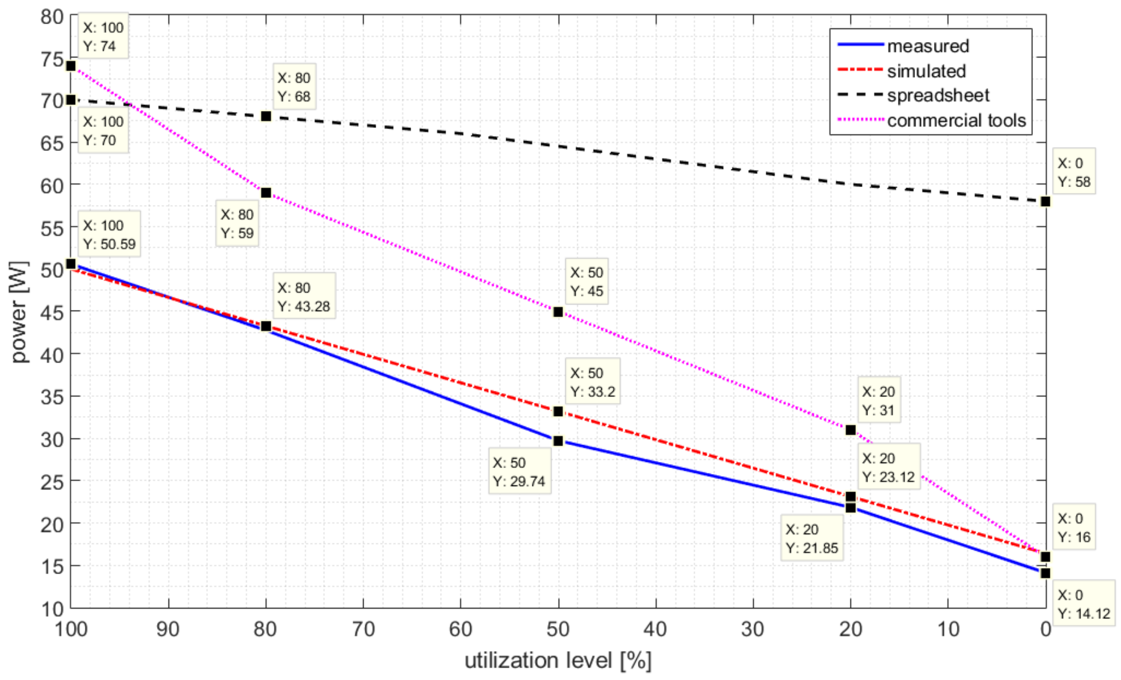


**Figure 140: Processor power consumption $(C18)$ –
an exemplary comparison between spreadsheet, commercial tools, measured and simulated power**

If we only consider the processor series *E5-2600* and their continuous development as a single static characteristic in our simulation model, we will provide the list of processors $(C7, C18, C3, C1)$ sorted by their peak power in ascending order. Figure 141 illustrates the simulation-based results in which the processor $(C18)$ has the lowest peak power because the thermal design power of $(C18)$ has the smallest value by $70W$. Consequently, we consider the thermal design power as an additional static characteristic and update the list $(C18, C7, C3, C1)$. In our approach, the simulation of the processor $(C3)$ is based on the simulation of the processor $(C1)$, which we proceed in the same manner accordingly with the processor $(C7)$ that relies upon the processor $(C3)$. We adjust the slopes and weight coefficients[397] regarding the significant characteristics to change the power correction of every processor. Furthermore, we observe the non-linear power consumption of the processor $(C1)$ because the amount of p-states is bigger than 15, as stated in the concept section. Appendix A3h provides the simulation-based results of processor power consumptions considering the processors $(C1 - C19)$.

[397] Slopes and weight coefficients: the slope of higher utilization levels is always larger than the slope at lower utilization levels $WF_{P_k}^{high} \gg WF_{P_k}^{low}$

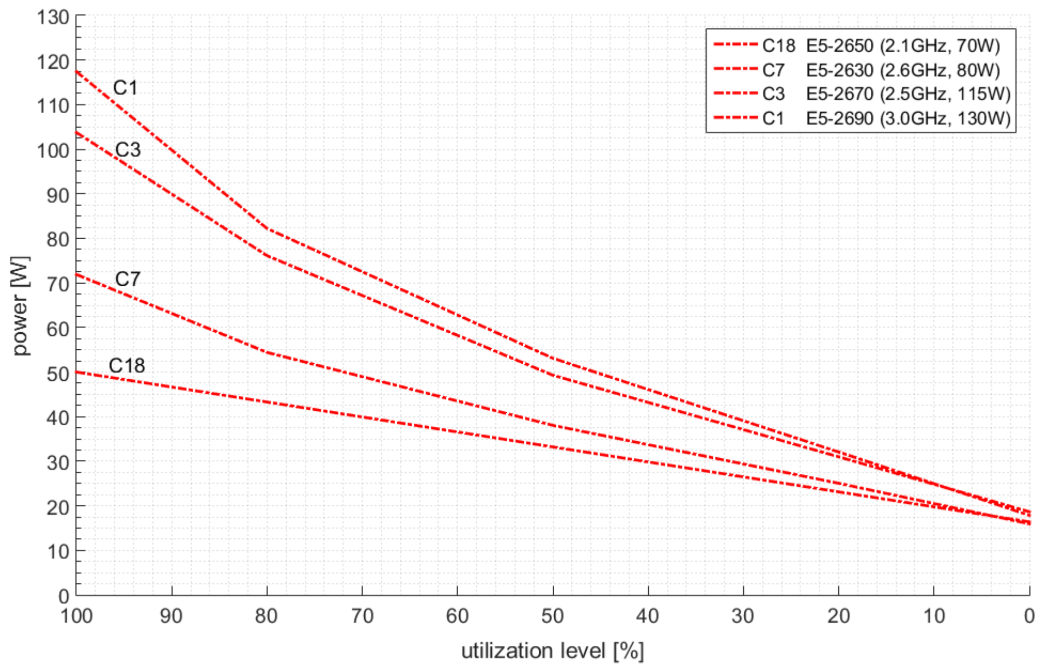


**Figure 141: Simulated processor power consumption** $(C1, C3, C7, C18)$

Table 81 lists the inaccuracy of our processor power simulation under the synthetic utilization levels regarding their *absolute* as well as *relative differences* (in the mean) between the simulated and measured values. Our aim is an error rate less than ten percent, which we specify as precise enough, see Chapter 4. We concentrate on the processors $(C1 - C19)$ of the *E5-2600v2* product family and observe the *absolute difference* in the range of $[-6.63, +6.63]W$. We can argue that our simulation-based results are reliable and adequate, because the inaccuracy is approximately $\pm 9.6\%$ when we neglect the results of the processors $(C2)$, $(C4)$, and $(C8)$. In case of these three processors, we observe the total *absolute difference* of $+4.34W$ up to $+9.2W$, especially at full utilization levels, which has a negative impact on the entire power simulation of the particular processor. The variance of the processor power increases with the processor generation, which is approximately $20\%$ up to $25\%$ measured from peak to peak [Fuj 2013]. The impact of process variation in the processor power consumption can be another reason for our inaccuracy [RLG et al. 2008].

**Table 81: Processor power accuracy – the simulated vs. the measured results**

| | Inaccuracy | |
|---|---|---|
| | *Absolute difference* [W] (mean) | *Relative difference* [%] (mean) |
| $C1$ | *+1.13* | *+3.6* |
| $C2$ | ***-6.63*** | ***-13.8*** |
| $C3$ | *+3.88* | *+7.6* |
| $C4$ | ***+6.63*** | ***+14.7*** |
| $C5$ | *-1.21* | *-9.6* |
| $C6$ | *+3* | *+3.2* |
| $C7$ | *+4.13* | *+9.6* |
| $C8$ | ***+5.37*** | ***+15.3*** |
| $C9$ | *+1.15* | *+1.4* |
| $C10$ | *-2.18* | *-8.2* |
| $C11$ | *+0.05* | *-1.4* |
| $C12$ | *+4.73* | *+5.2* |
| $C13$ | *+5.69* | *+2.1* |
| $C14$ | *+4.25* | *+0.3* |
| $C15$ | *+1.78* | *+0.6* |
| $C16$ | *-0.63* | *-1.0* |
| $C17$ | *-1.54* | *-0.9* |
| $C18$ | *+1.39* | *+6.7* |
| $C19$ | *-2.75* | *-8.2* |

A sufficiently precise simulation requires static and dynamic characteristics to be universally applicable on future and uncertain systems that we demonstrate by our processor-specific power values considering the synthetic utilization levels. Nevertheless, we can predict future components in our simulation model that follow the same trend as the processors before.

### 7.3.5 Conclusion

We demonstrate that our simulation model at a high level of abstraction is sufficiently precise (less than $+10\%$ *relative difference*) when we consider the relevant static and dynamic characteristics of the memory modules and processors. Our simulation results are more accurate in comparison to the results gained from commercial tools or defined by spreadsheets. We reduce the over-estimation of the worst-case power, e.g. the peak power at the full utilization level, and decrease the power gap at lower utilization levels when we compare our simulation results to spreadsheets. We improve the estimation process, especially at lower utilization levels, so that we can define the power consumption in an early design phase more precisely. We simulate equivalent results concerning precision and accuracy in comparison to academic approaches, taking into account that we investigate less effort to analyze and measure each component, such as various memory modules. The authors of [Han 2007] analyze the specific frequencies executing different benchmarks, but we show in

Section 7.2.4 that the various processor-intensive micro-benchmarks have little impact on the peak power consumption. We consequently assume that the processor frequencies rely upon the utilization level. In the work of [Han 2007], the authors state that the power linearly depends upon clock throttling. As a result, we assume that the processor power is linear to the certain p-state (voltage-frequency pair). In contrast to the work in [GFN et al. 2006], we specify our components by the relevant technical specification $CH_{TS}$, static $CH_{CFG}^{ST}$ and dynamic characteristics $CH_{CFG}^{DY}$ wherein we neglect the instruction-level details, as analyzed in [BGM et al. 2010] by the access rate. We simulate the entire processor power dependent on the cache sizes $(L2, L3)$, which is defined as core-specific power consumptions in the power models in [RRK 2008, BM 2012 KJC et al. 2014]. In fact, the amount of processor cores is an additional relevant characteristic of processor power consumption. Our simulation model flexibly reacts upon the characteristic changes of the components, which finally reduces the measurement effort of the particular component series. We predict future components, but assume a continuous development of the components, wherein we adjust the certain weight coefficients upon the basis of the specific characteristics and technology changes. In the work of [Han 2007], the authors define a processor model of the *Intel Pentium M 755*[398] generation, which we do not include in our simulation model because the processor is a desktop processor, while we concentrate upon server-specific processors. The authors in [Riv 2008] define the power model of the server processor *Intel Xeon 5130*[399] similar to the processor model in [THS 2010] suitable for an *Intel Xeon E5430*[400] processor. Both are three processor generations (*Core* generation) away from our SUT-specific processor generation[401], called *Ivy Bridge*. The authors in [TDM 2011] analyze an *Intel Xeon E5540*[402] processor, including the *Intel Xeon Nehalem* architecture, which is two processor generations away from our SUT processor generation. We do not support the *Intel Pentium M (M 755), Intel Xeon Core (5130, E5430),* or *Nehalem (E5540)* architecture in our simulation model, because we want to limit the model complexity. Our simulation model is actually limited to the server-specific generations of the *Intel Xeon* processors, in particular, *Sandy Bridge*, *Ivy Bridge*, and *Haswell*, because the experimental results of Fujitsu are not accessible to the public and are restricted to these three *Intel Xeon* generations. In general, we need to adjust the processor generations and estimate the generation dependencies.

[398] Intel Pentium M 755: (Pentium M, Dothan), 2.0GHz base frequency, 7.5W TDP, one core, 2MB L2 cache, 400MHz

[399] Intel Xeon 5130: (Core, Woodcrest), 2.0GHz base frequency, 65W TDP, two cores, 4MB L2 cache, 1333MHz

[400] Intel Xeon E5430: Xeon (Core, Harpertown), 2.66GHz base frequency, 80W TDP, four cores, turbo, 6MB L2 cache, 12MB L2 cache, 1333MHz

[401] Processor generation: https://ark.intel.com/#@Processors, https://en.wikipedia.org/wiki/List_of_Intel_Xeon_microprocessors

[402] Intel Xeon E5540: Xeon (Nehalem, Gainestown), 2.53GHz base frequency, 80W TDP, four cores / eight threads (4C/8T), hyper-threading, turbo, 8x256KB L2 cache, 8MB L3 cache (TLC), 5.86GT/s QPI, 1066MHz

## 7.4 Improvement Analysis regarding Server System Optimization

### 7.4.1 Objective

Sections 7.2 and 7.3 evaluate the calculation methods on the basis of isolated components considering the heterogeneous workload scenarios and flexible changes of the component characteristics. On the basis of the evaluation results in Sections 7.2 and 7.3, we can trust in the accuracy of the aspect-based calculation methods concerning the variable adjustments. Our aim of this section is to evaluate the server system optimization in which we alternate the technical specification $CH_{TS}$, static characteristic $CH_{CFG}^{ST}$, or dynamic characteristics $CH_{CFG}^{DY}$ as part of the decision variables in our optimization strategy. We simulate the entire server system and analyze the improvements regarding our adjustments. For this purpose, we observe the necessary performance scores that we will consider in our performance models and that will finally become part of our *energy efficiency analysis.* In our empirical analysis, we concentrate on the server system architecture of our rack-based system under test. Additionally, our complete server simulation shall improve the energy efficiency concerning the peak performance, accordingly, reduce the energy consumption associated with the decrease of the heat dissipation, and consider a realistic workload scenario instead of a worst-case scenario. In contrast to [ERK 2006, RRK 2008, Ran 2010], we want to optimize the server system for low utilization because the average server utilization is less than ten percent and always lower than 50 percent in a data center [KFK 2008]. The server optimization offers a high level of potential savings in the energy consumption and total costs. A couple of alternative approaches[403] exist that overcome low utilization in data centers. If these techniques are applied, we would be able to optimize a server system for high utilization as well.

### 7.4.2 Evaluation Criteria

At first, our server system model rapidly needs to react on the workload scenarios (category-specific utilization levels) provided as external *stimuli*. Secondly, it has to react precisely to the characteristic changes that occurred in the internal *system-wide optimization engine* in our simulation. We assume that we can optimize the server system in our step-based analysis, evaluate the results, and find the possible impacts of our optimization. In this analysis, we answer the following questions:

- How much can we optimize the energy efficiency by adjusting a more suitable configuration or characteristic?
- How much amount of power consumption[404] does an improved server system (configuration or characteristic) accomplish regarding a specific workload scenario?

[403] Alternative approaches: e.g. virtualization, scheduling, or allocation techniques
[404] Power consumption: associated to the carbon footprint

As a prerequisite, we assume that it is possible to improve the energy efficiency of the server system at any point of time. Further prerequisites are that we can evaluate and alternate the components as a local solution (step-base) during the simulation. We assume that the operating point of a server system is only a local optimization for recent management techniques or optimization strategies at the specific time. A pre-assumption of our simulation is that the component-based models consider the power and temperatures of our calculation-based methods, but include the results gained from the Fujitsu-specific performance measurements. Our evaluation criteria are the *absolute* and *relative differences* between the initial server system configuration (SUT) and our optimized configuration concerning the local solution and the global optimal solution.

### 7.4.3 Experimental Setup

We specify the general evaluation environment and measurement infrastructure in Sections 7.1.1, 7.1.2, and 7.1.3. In this section, we exclusively study our system under test[405] and specify the experimental setup of the *energy efficiency analysis*. In contrast to Sections 7.2 and 7.3, we simulate the entire server system and apply our hierarchical approach in the simulation model. Figure 142 presents the corresponding block diagram of the controller layer in which we visualize, analyze, evaluate, and optimize the server system.

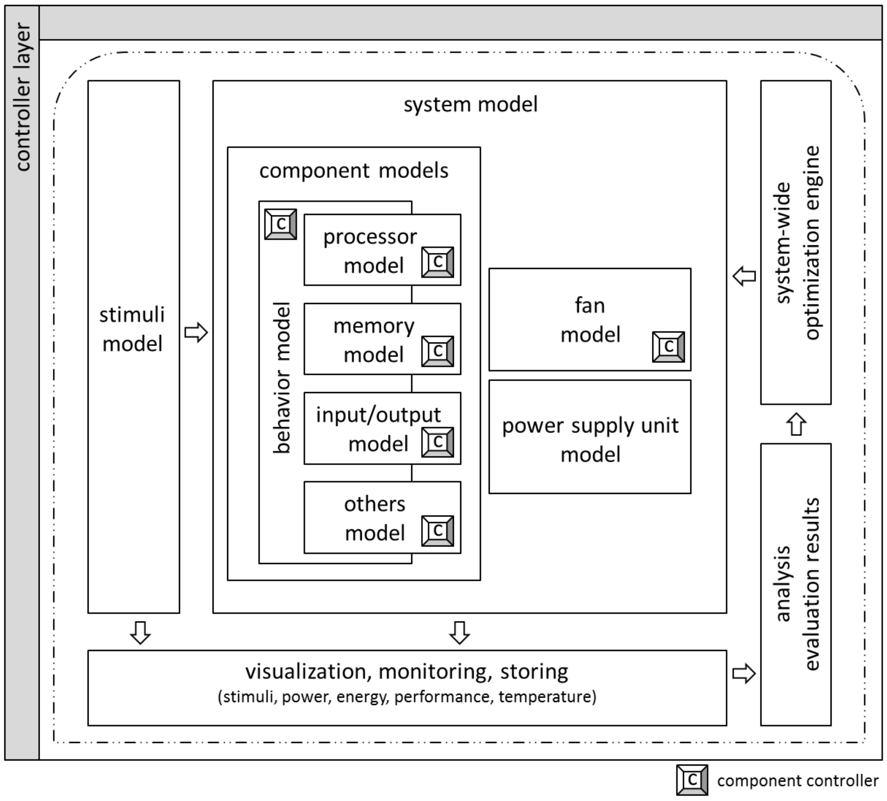


Figure 142: Controller layer – block diagram considering the entire system model for evaluation purpose

[405] System under test: described in detail in Section 7.1.1

We connect the *component models* in the *system model* to enable the communication and the corresponding effects of the *behavior models*. Nevertheless, we activate the particular *calc_category()* method within each component model, as described in Section 7.2. We enable the *system-wide optimization engine* because we analyze the results of the entire system model to optimize the energy efficiency. We analyze the possible impacts of several characteristics on the respective aspects and decide on the suitable alternation strategy. We apply our optimization and alternation strategy in the *system-wide optimization engine* in which we adjust and restrict the set of decision variables to reduce the complexity and simulation effort, as described in detail in Section 6.3.

In this evaluation section, we externally generate the *stimuli* that distribute all component-specific utilization levels to our *system model* and its integrated component-based models, as shown in Figure 143. Our simulation model reacts to either a steady or continuous workload scenario, which together with the ambient temperature is configurable in the graphical user interface. We can import the utilization levels of the real-life measurements as an input parameter or generate synthetic workload scenarios.

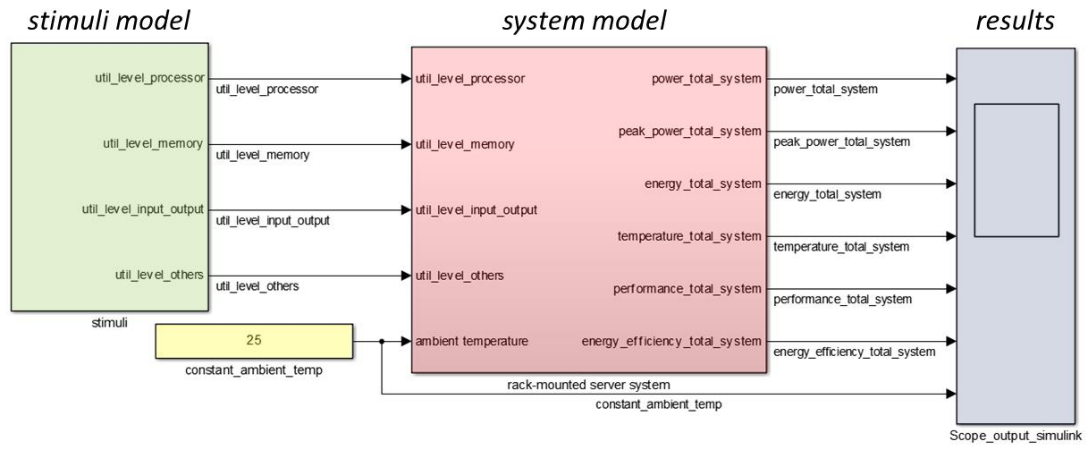


**Figure 143: Simulation model – stimuli and server system considering the entire system model**

### 7.4.4 Results and Analysis

The foundations of our optimization analysis consists of the total memory, processor, and system-specific performance scores that we study while executing the *PassMark Memory*, *PassMark CPU*, and *SPECpower* benchmarks. In our simulation model, we create a database to specify the particular component-based as well as system-specific performance scores concerning their various characteristics and benchmarks. Furthermore, we include the relative performance scaling based on our measurements, such as adding an extra processor or extending the memory capacity. Appendix A3i provides a brief overview of the performance scores and findings related to our system under test.

In the following section, we present the potential for improving the energy efficiency of the processor, memory, and entire system on the basis of theoretical considerations. We especially analyze the performance-to-power ratios in relation to the adjustments of the server configuration that we gained while executing diverse benchmarks. Afterwards, we exemplarily show the performance-to-power ratios of the system under test $\theta_C$ resulting from the *SPECpower* benchmark that we consider as base energy efficiency $EE_{BASE}$. We present the alternative configurations and their corresponding energy efficiency $EE(\theta_C^l, \vec{U}_{t_k})$ resulting from our step-based analysis in our optimization. Moreover, we exemplarily describe how we observe the local set of solutions (Pareto front) that satisfy the objective functions and illustrate the results of our global analysis, specified as post-process, in which we reduce the local set of solutions to present the global optimal solution. Finally, we compare the results of our optimization strategy concerning the base energy efficiency and energy consumption.

***Memory, Processor, and System Performance-to-Power Ratios***

In parallel to the memory and processor evaluation (7.2 and 7.3), the particular benchmarks store the performance score that we consider when we simulate the energy efficiency. Figure 144 and Figure 145 exemplarily show the performance-to-power ratios (on top of the bars) as the results of the *PassMark CPU* $(PCx.y.z)$, the three bars on the left and the *PassMark Memory* $(PMx.y.z)$ benchmarks the three bars on the right of the figures. It can be observed that if we double the memory capacity in a server system with an exclusive processor $(x = 1)$, either in a single memory module $(\hat{z} = z * 2)$ or as an additional memory module $(\hat{y} = y * 2)$, the performance-to-power ratio will approximately increase by $[2.0, 11.5]\%$ at $(PC1.y.z)$ and by $[52.8, 106.9]\%$ at $(PM1.y.z)$, see Figure 144. If the system has two processors, the performance-to-power ratio of the processor increases at the *PassMark CPU* benchmark $(PC2.y.z)$ by $[5.3, 24.1]\%$ when we double the memory capacity. In contrast, the performance-to-power ratio of the memory modules decreases by nearly $-5.1\%$ when we double the memory capacity of the single module, but increases by around $80.1\%$ at the *PassMark Memory* benchmark $(PM2.y.z)$ when adding an extra memory module, as shown in Figure 145.

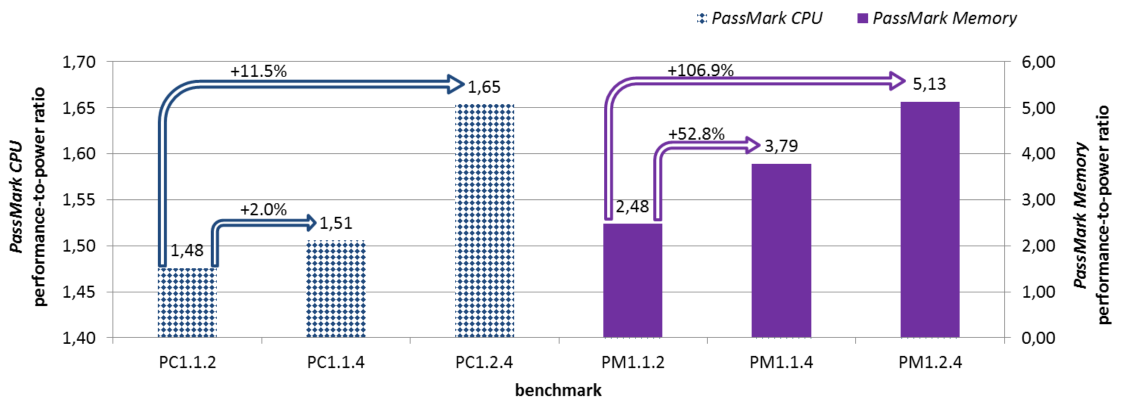


**Figure 144:** ***PassMark CPU*** **$(PC1.y.z)$ and *PassMark Memory* $(PM1.y.z)$ performance-to-power ratios**

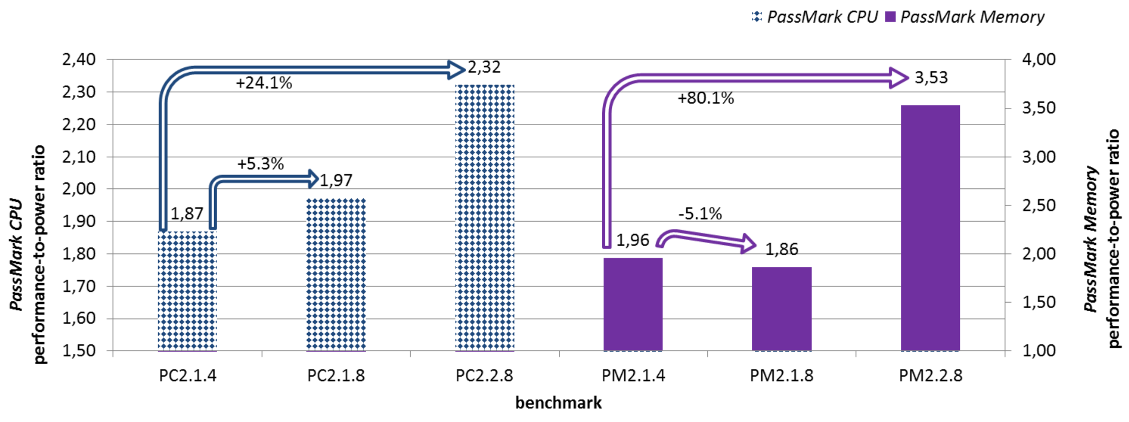


**Figure 145:** ***PassMark CPU*** **$(PC2.y.z)$ and *PassMark Memory* $(PM2.y.z)$ performance-to-power ratios**

Furthermore, we observe that an additional processor and its related second memory module[406] have a significant impact of approximately $[-50.9, -21.0]\%$ on the performance-to-power ratio of the memory modules, as shown in the *PassMark Memory* benchmarks, see Figure 146. In contrast, when the server system executes the *PassMark CPU* $(PCx.y.z)$ benchmarks in which we provide an additional processor and memory module, we observe improvements of the performance-to-power ratios of the processor by approximately $26.4\%$, $30.5\%$, and $40.6\%$, as shown in Figure 147.

[406] Second memory module: server system settings require a memory module per processor, based upon the fact of the regular expansion

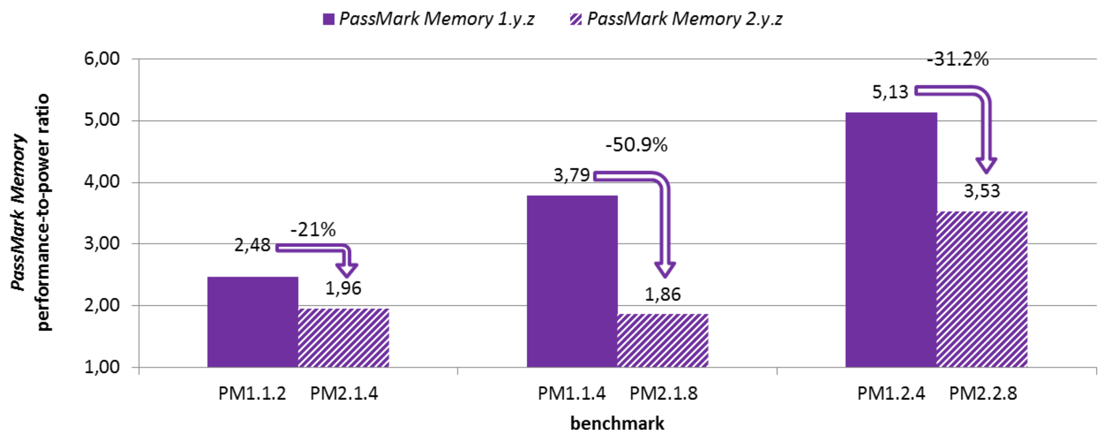


**Figure 146: *PassMark Memory* performance-to-power ratios**$(\hat{x} = x * 2)$

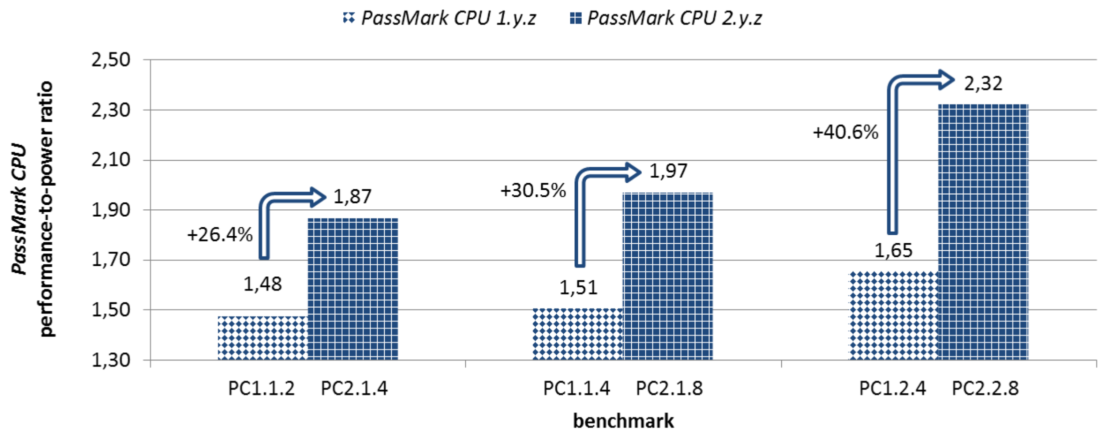


**Figure 147: *PassMark CPU* performance-to-power ratios**$(\hat{x} = x * 2)$

The authors of [DEP et al. 2009] present the typical performance gains [1.5,1.7] when doubling the amount of the processor and the memory capacity of an *IBM x3850 M2* server[407]. The theoretical potential improvements of the processor-specific and memory-specific energy efficiency lead to the assumption that we can easily improve the energy efficiency of the entire system. We investigate and clarify the actual performance-to-power ratios by evaluating our real server system.

[407] *IBM x3850 M2*: http://www-07.ibm.com/systems/includes/content/x/hardware/enterprise/x3850m2/pdf/xso03033usen.pdf

***Server System Optimization***

We exemplarily optimize the energy efficiency of the system under test, as specified in Table 47, because we can easily compare our simulation results with the measurement-based data. In our optimization strategy, we adjust the dynamic characteristics, static characteristics, and technical specifications relying upon the actual thermal dissipation and power consumption of the single components. The optimized components will probably have a positive effect on the total energy efficiency of the entire server system. In the following sections, we concentrate on the potential local and global optimal solutions for the energy-efficient server configuration, irrespective of the optimization phase[408] in which the solutions are found.

We exemplarily execute the trace of the *SPECpower* benchmark ($SP1.2.8$) (gained by the *Intel Power Thermal Utility*) in our simulation considering the component-specific utilization levels in the period of $T = [0{,}3732]s$ presented by their target throughputs of the system between $[10{,}100]\%$, as shown in Figure 148. At first, we check the plausibility of the normalized performance-to-power ratios and secondly, we analyze the local and global optimal solutions.

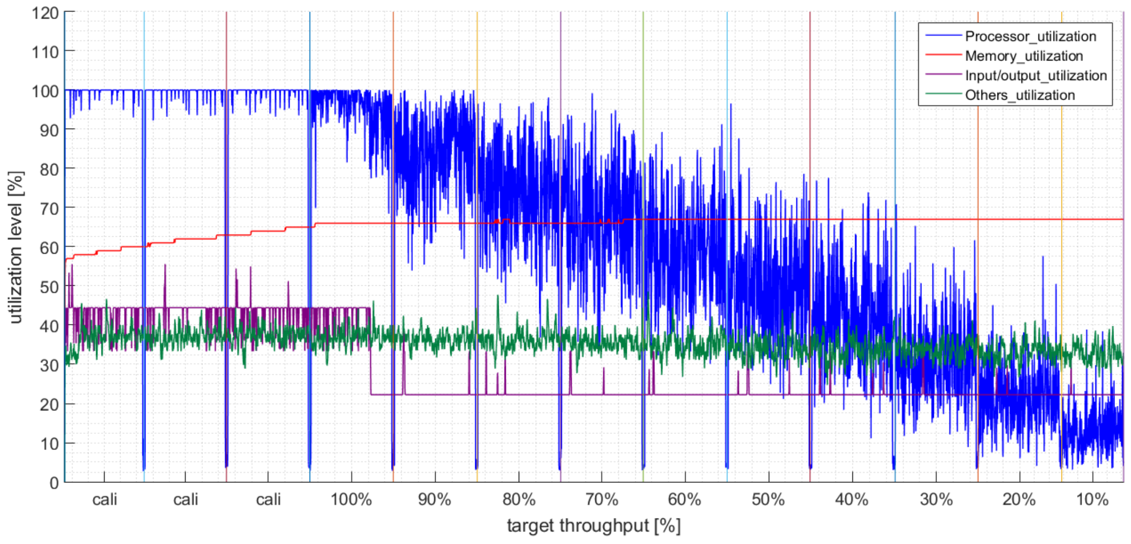


**Figure 148: Component-specific utilization level at *SPECpower***

[408] Optimization phase: primary or secondary phase of alternate decision variables

The processor-intensive workload (solid blue line) produces up to $70\%$ target throughput the significant part of the total power consumption, see Figure 149. The power consumption of the others (solid green line) constitutes the motherboard and fans, which we can partly influence in our optimization strategy. We neglect the I/O-based power (purple solid line) in the case of a processor-bounded or memory-bounded workload, as it has little influence on the entire power dissipation and consider them as static power. In contrast, the memory utilization (solid red line) highly depends upon the utilization level, but in our example, the memory power is the lowest part of the total power consumption. Accordingly, we concentrate upon the processor optimization.

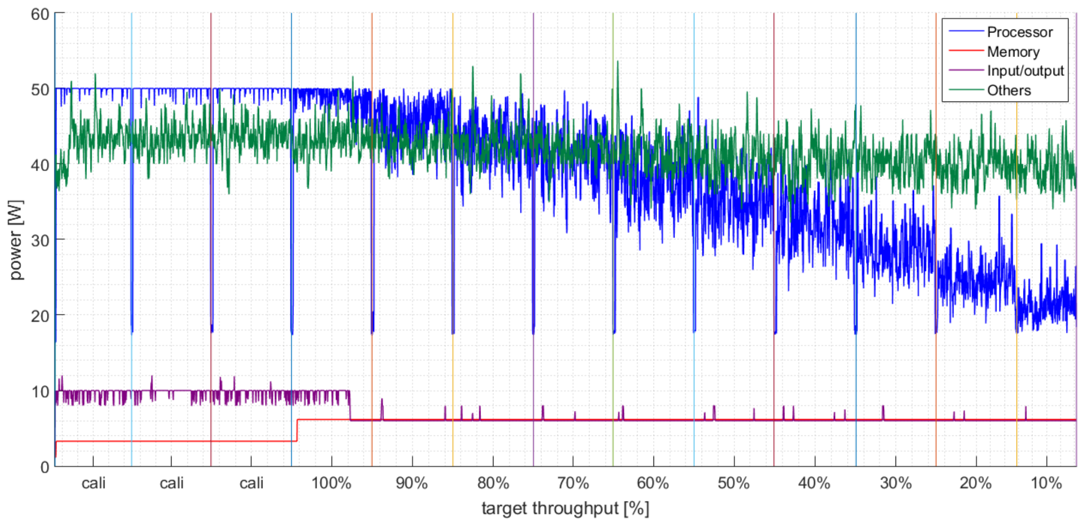


**Figure 149: Component-specific power consumption at *SPECpower***

Our measurements do not consider the performance of the input/output and other components. Nevertheless, Figure 150 presents the normalized performance-to-power ratios of the processor (solid blue line), the memory (solid red line), and the entire system under test $\theta_C$ (solid magenta line) that we consider as base energy efficiency $EE_{BASE}$ in our optimization strategy. We observe that the performance-to-power ratio of the memory modules is nearly stable around $0.81$ from $100\%$ down to $0\%$, but increases during the calibration phase from $1.3$ up to $1.5$. The performance-to-power ratio of the processor is in linear proportion to the processor utilization level. The performance-to-power ratio of the system increases from $1$ up to $1.4$ until the target throughput of $40\%$ and decreases at lower target throughputs.

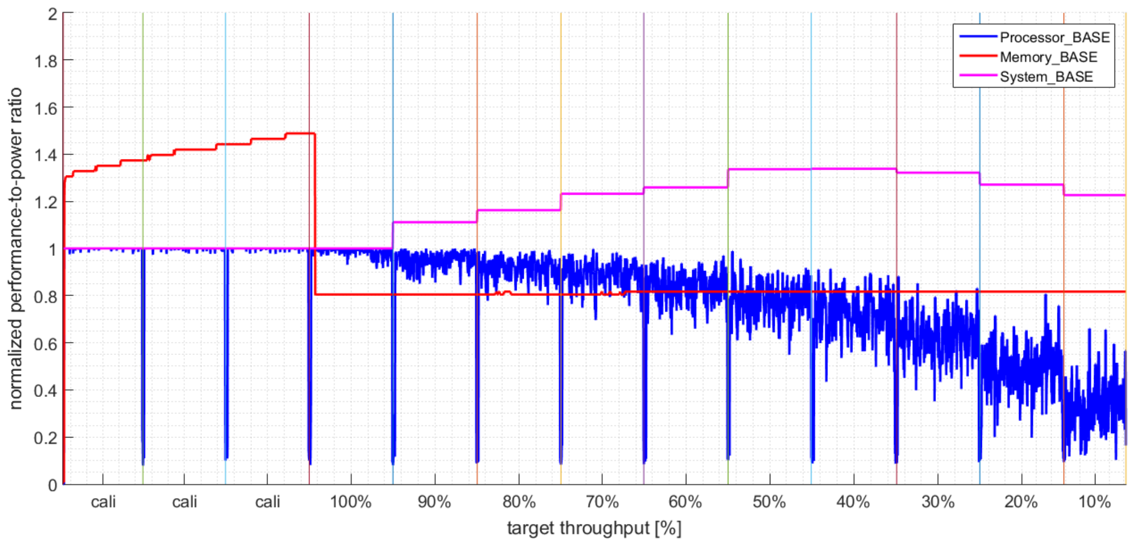


**Figure 150: Normalized performance-to-power ratios at *SPECpower***

Our aim is a server system optimization regarding energy efficiency (performance-to-power ratio) that simultaneously results in the mutually contradicting objectives: minimizing the power and maximizing the performance. We exemplarily optimize the processor in the simulation in which we execute the step-based analysis at each discrete time $t_k$ of $T$ and iterate all possible decision variables to specify various alternative processor configurations. Figure 151 shows all alternative processor configurations (many-colored crosses) as the results of our step-based energy efficiency analysis in the optimization process considering the entire simulation period $T = [0{,}3732]s$ on the x-axis and the normalized performance-to-power ratio $EE(\theta_C^l, \vec{U}_{t_k})$ on the y-axis, which we further analyze in the post-process to find the Pareto front. In the figure, we simply identify the gaps of the processor utilization levels around $4\%$ as a set of vertically aligned alternative processor configurations in regular intervals containing the normalized performance-to-power ratios between $0$ and $0.6$. We observe that the number of alternative processor configurations (many-colored crosses) differ with each time, especially when we analyze the period $T = [0{,}1400]s$. The increasing number of alternative processor configurations in the continuous optimization process will increase the simulation time, because every alternative processor configuration activates the aspect-based calculation methods at each discrete time $t_k$.

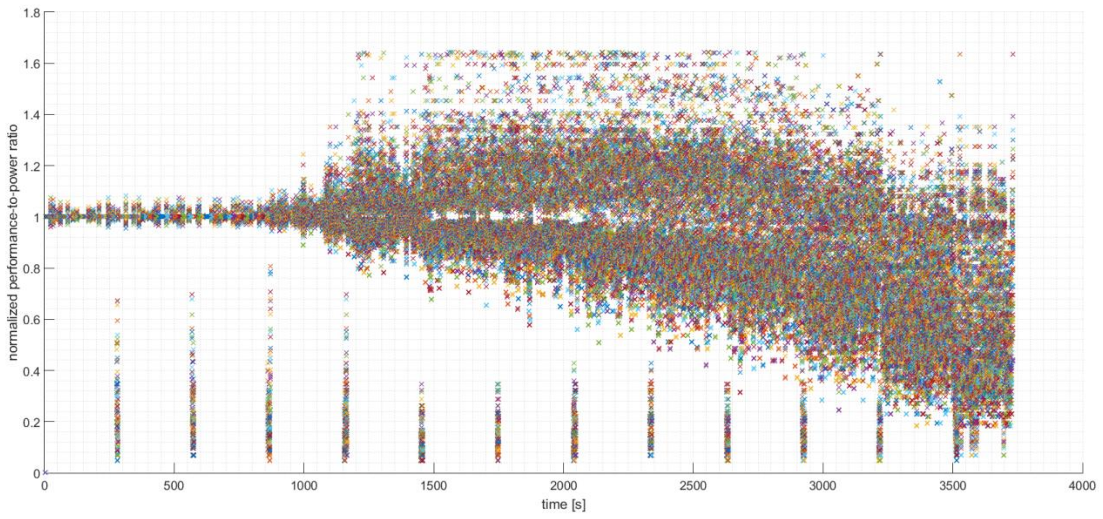


**Figure 151: Alternative processor configurations (normalized performance-to-power ratio) at *SPECpower* $T = [0, 3732]s$**

We backwards transform Figure 151 into a representation concerning the normalized power and normalized performance. Figure 152 presents the alternative processor configurations (many-colored crosses) in a 3-dimensional representation in which the x-axis shows the entire period $T = [0,3732]s$, the y-axis specifies the normalized power, and the z-axis defines the normalized performance. This figure gives a rough impression of the optimizations' complexity on the basis of the alternative processor configurations (many-colored crosses), respectively their numbers.

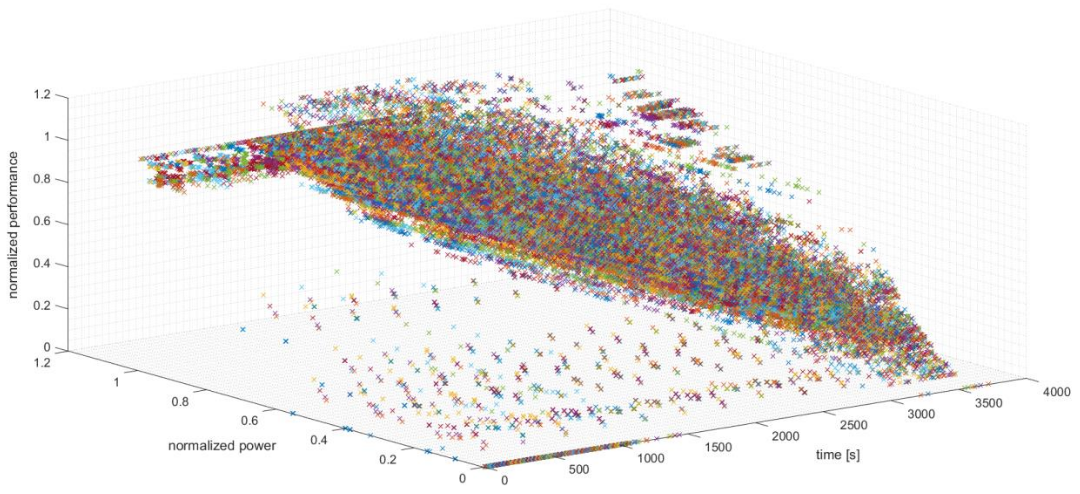


**Figure 152: Alternative processor configurations (normalized power, normalized performance) at *SPECpower* $T = [0, 3732]s$**

We simplify the graph of Figure 152 by limiting the period $T = [0,1400]s$ on the x-axis to show an easier-to-read representation of the alternative processor configurations resulting from our optimization process, which we further store in the ranked lists in descending order. We observe in Figure 153 that the number of possible alternative processor configurations (many-colored crosses) is less when the processor is fully utilized, as shown in the beginning $T = [0,876]s$ in comparison to the period $T = [876,1400]s$ when the processor utilization decreases. The varying number of alternative processor configurations results from the optimization and the alternation process considering the weighted dynamic characteristics, static characteristics, and technical specifications to fulfill the mutually exclusive objectives, such as power and performance conditions at the specific workload.

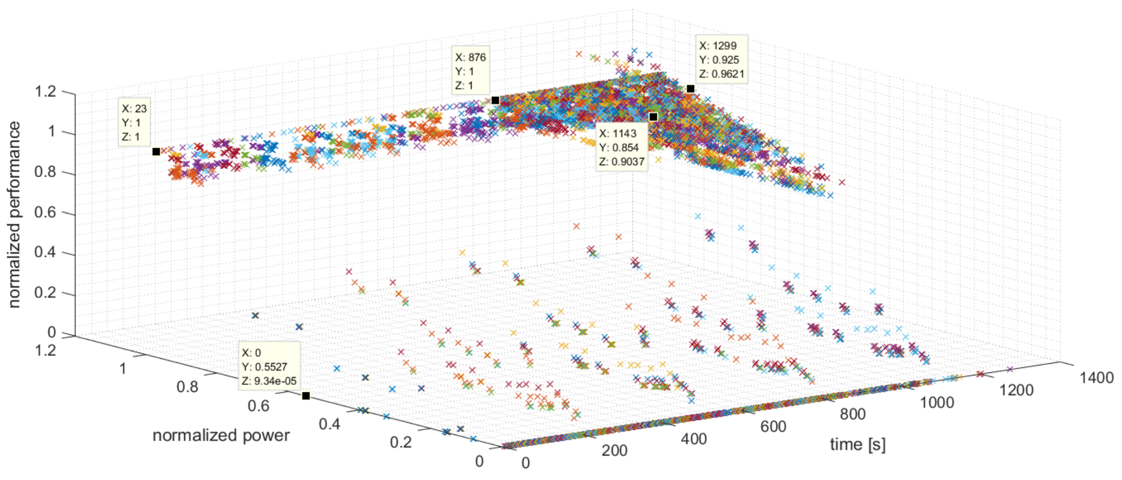


**Figure 153: Alternative processor configurations (normalized power, normalized performance) at *SPECpower* $T = [0, 1400]s$**

We exemplarily analyze the large number of processor-specific performance-to-power ratios (Figure 151) to find the local set of optimal solutions (Pareto front) at each time. The following steps pertain to our global analysis, which we separately execute as a post-process of our simulation.

In our example, we present the alternative processor configurations of the optimization process at the times $t_1 = 395s$, $t_2 = 577s$, $t_3 = 876s$, $t_4 = 1143s$, and $t_5 = 1299s$ and analyze them to check whether we have improved the energy efficiency of the base processor configuration[409]. Therefore, we analyze the actual utilization level, the corresponding normalized power $\frac{PO_C}{\max(PO_C)}$, the normalized performance $\frac{PE_C}{\max(PE_C)}$, and the normalized performance-to-power ratio $\frac{G_1^C(EE)}{\max\left(G_1^C(EE)\right)}$ of the base processor configuration, as listed in Table 82.

[409] Base configuration: the initial configuration $\theta_C$ specifies the base energy efficiency $EE_{BASE}$ (normalized performance-to-power ratio)

Table 82: Base processor data at system under test (time, utilization level, normalized power, normalized performance, and normalized performance-to-power ratio)

| Time | Utilization level [%] | Normalized power | Normalized performance | Normalized performance-to-power ratio |
|---|---|---|---|---|
| $\boldsymbol{t_1 = 395s}$ | 100 | 1 | 1 | 1 |
| $\boldsymbol{t_2 = 577s}$ | 4 | 0.356 | 0.041 | 0.115 |
| $\boldsymbol{t_3 = 876s}$ | 97 | 0.978 | 0.968 | 0.989 |
| $\boldsymbol{t_4 = 1143s}$ | 90 | 0.935 | 0.903 | 0.966 |
| $\boldsymbol{t_5 = 1299s}$ | 78 | 0.854 | 0.782 | 0.916 |

In our optimization strategy, we specify the alternative processor configurations that have a lower normalized power or a higher normalized performance in comparison to the base configuration (without optimization), see Table 82. Figure 154 exemplarily presents the alternative processor configurations, marked by the many-colored crosses[410] in the subplots at the times $t_1 = 395s$, $t_2 = 577s$, $t_4 = 1143s$, and $t_5 = 1299s$ with the normalized power on the x-axis and the normalized performance on the y-axis. In our optimization strategy, we analyze all alternative processor configurations at each discrete time $t_k$ to find a local set of solutions (Pareto front) that satisfy the mutually contradicting objectives of the energy efficiency. Consequently, in Figure 154 we present the local set of optimal solutions (Pareto points) of the processor configurations, which we identify within the post-process, marked with red dots[411].

[410] Many-colored crosses: alternative processor configurations do not always have the same colors, due to the restricted color representation and confusable colors display
[411] Red dots: specify various alternative processor configurations

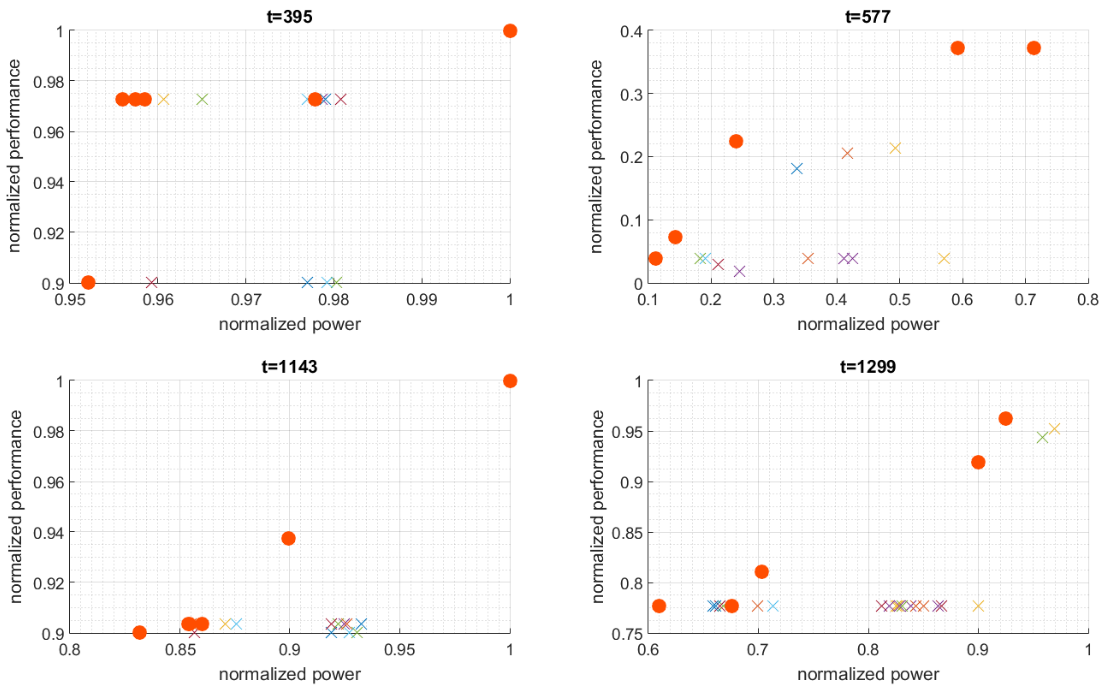


**Figure 154: Exemplary processor optimization (normalized power, normalized performance, Pareto front) at *SPECpower* $t_1 = 395s, t_2 = 577s, t_4 = 1143s, t_5 = 1299s$**

The local set of optimal solutions (represented as red dots) has a higher performance-to-power ratio in comparison to the base processor configuration. Figure 155 presents the identical simulation results of the alternative processor configurations (many-colored crosses) and the local set of optimal solutions (red dots) at the times $t_1 = 395s$, $t_2 = 577s$, $t_4 = 1143s$, and $t_5 = 1299s$, presenting the related performance-to-power ratios on the y-axis. The Pareto points present the ideal short-term solutions of our optimization process.

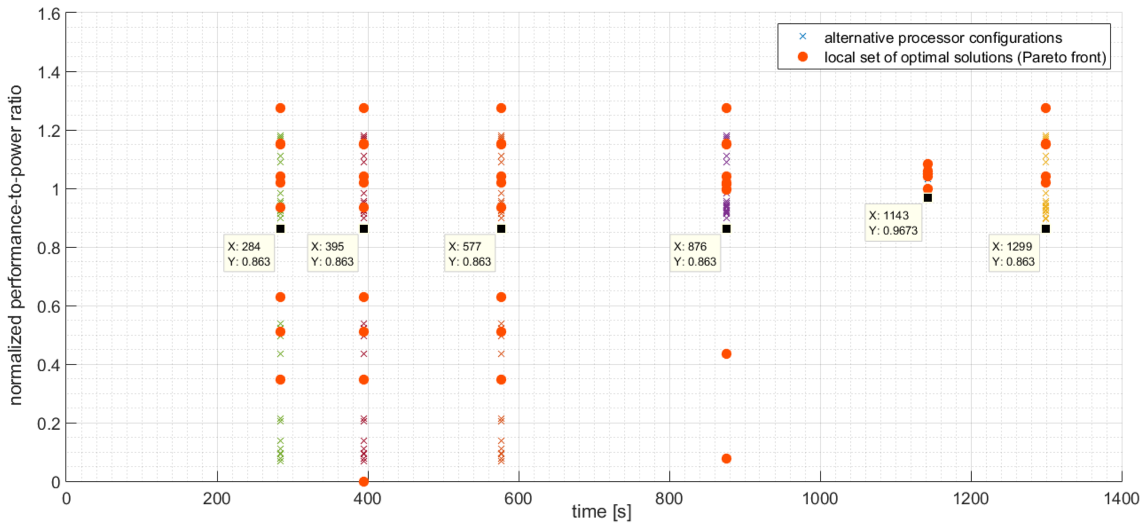


**Figure 155: Exemplary processor optimization (normalized performance-to-power ratio) at *SPECpower* $t_1 = 395s, t_2 = 577s, t_3 = 876s, t_4 = 1143s, t_5 = 1299s$**

In the post-process (global analysis), we search the global optimal solution as a long-term optimum over the period of time. We consider all local optimal solutions, which involves 108 unique processor configurations, in our exemplary optimization. The globally optimal solution dominates for the longest period of time and represents a rough approximation of an intuitive selection, as shown in Table 83. To be more precise, we additionally consider the balance of the power and respective energy in the integrand of the time integral that leads to the global optimal processor configuration in Table 84.

**Table 83: Intuitive global optimal processor configuration**

| Processor | Intuitive global optimal configuration |
|---|---|
| Family (Series) | *Intel Xeon E5-2643 v2* $(C14)$ |
| Generation | Ivy Bridge EP (Romley) |
| Frequency | 3.5GHz |
| Hyper-threading / turbo | Enabled / enabled |
| Enabled | 6 cores, 2 chips |
| Hardware threads | 12 (2 / core) |
| L1 Cache | 8x32KB instruction caches, 8x32KB data caches |
| L2 Cache | 6x256KB |
| L3 Cache | 25600KB |
| Thermal design power (TDP) | 130W |

Table 84: Global optimal processor configuration considering the power and respective energy

| Processor | Global optimal configuration considering the power and respective energy at the time |
|---|---|
| Family (Series) | *Intel Xeon E5-2603 v2* $(C10)$ |
| Generation | Ivy Bridge EP (Romley) |
| Frequency | 1.80GHz |
| Hyper-threading / turbo | Enabled / enabled |
| Enabled | 4 cores, 2 chip |
| Hardware threads | 4 (2 / core) |
| L1 Cache | 8x32KB instruction caches, 8x32KB data caches |
| L2 Cache | 4x256KB |
| L3 Cache | 10280KB |
| Thermal design power (TDP) | 80W |

Table 85: Global optimal processor configuration concerning the performance-to-power ratio

| Processor | Global optimal configuration concerning the performance-to-power ratio |
|---|---|
| Family (Series) | *Intel Xeon E5-2637 v2* $(C15)$ |
| Generation | Ivy Bridge EP (Romley) |
| Frequency | 3.5GHz |
| Hyper-threading / turbo | Enabled / enabled |
| Enabled | 4 cores, 2 chips |
| Hardware threads | 8 (2 / core) |
| L1 Cache | 8x32KB instruction caches, 8x32KB data caches |
| L2 Cache | 4x256KB |
| L3 Cache | 15360KB |
| Thermal design power (TDP) | 130W |

The top graph in Figure 156 exemplary presents the trace of the processor-specific utilization levels (by the *Intel Power Thermal Utility*) that we gained in the *SPECpower* benchmark, in which we consider the same intervals marked by the vertical solid lines and horizontal axes (target throughput) as in Figure 150. In the middle of the graph, we present the simulated power consumptions $[W]$ of the base configuration (solid blue line), and the globally optimal processor described in Table 84 (dashed-dotted red line). The lower graph of the figure presents the relative power optimization of the processor as a purple solid line. We observe the highest relative power optimization of approximately $85.2\%$ when we fully utilize the processor. The exemplary presented processor configuration provides the most energy reduction in the range of $[13.8,85.2]\%$, which is in a mean of $53.3\%$. In contrast, the intuitive global optimal solution reduces the power only by a mean value of nearly $5.2\%$.

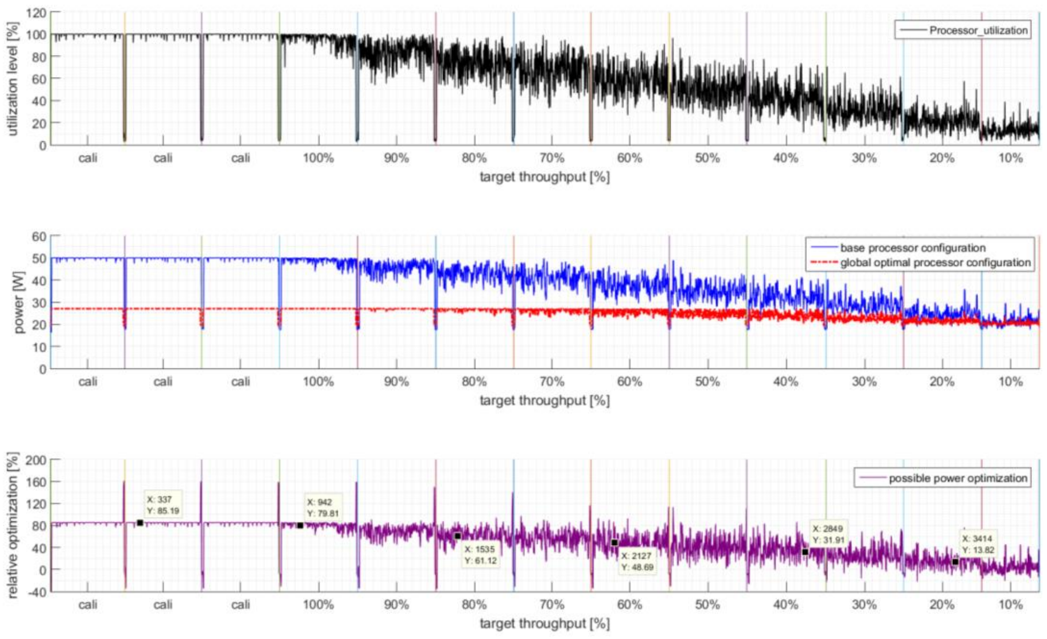


**Figure 156: Relative power optimization of the processor (Table 84)**

Finally, we compare the results of our optimization strategy concerning the base energy efficiency. In Figure 157, we concentrate on the normalized performance-to-power ratio of the base processor configuration (solid blue line), as shown in the middle of the graph. We observe an increasing relative optimization of the processor (Table 85) concerning the performance-to-power ratio, presented as a purple solid line in the lower graph. In our example, the globally optimal solution has a higher impact on the lower utilization levels in comparison to the higher utilization levels. We observe the relative optimization in the range of $[\pm 0{,}88.2]\%$, which is in a mean of $12.2\%$. Our relative improvement of the performance-to-power ratio is higher than the improvements gained by an additional memory module or doubled memory capacity in a single processor configuration $(11.5\%\ at\ PC1.y.z)$ while executing a processor-bounded workload. Our optimization process neglects an extra processor in the local set of solutions, which may improve the performance-to-power ratio of nearly $40.6\%$, but the additional processor consumes more energy over time and is not part of the Pareto points.

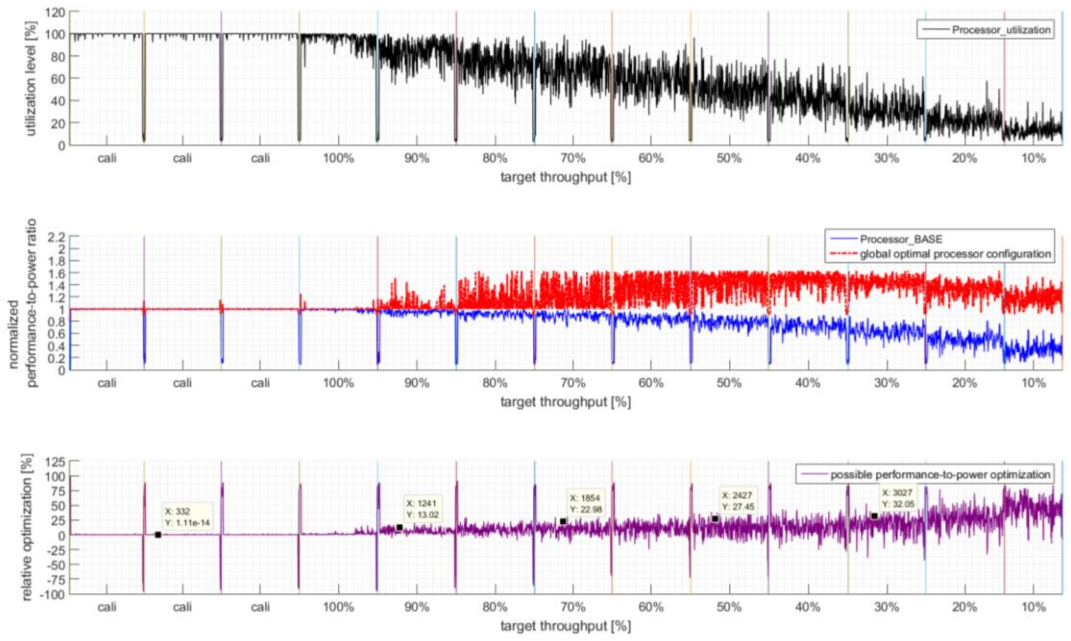


**Figure 157: Relative optimization of the performance-to-power ratio – processor (Table 85)**

Finally, we observe an absolute mean power reduction of approximately $17W$ considering the entire server system.

## 7.5 Summary

To evaluate our multi-aspect full-system server model, we present a series of experimental analyses, specifically: *Accuracy Analysis*, *Impact on Characteristics Changes*, and *Energy Efficiency Analysis*. We develop a prototypical simulation model, which is actually limited to the server-specific generations of the *Intel Xeon* processors - primarily, *Sandy Bridge*, *Ivy Bridge*, and *Haswell* - to limit the model complexity. In our simulation, we consider a mean usage of the subsystems and components specified as category-specific utilization levels to guarantee the compatibility with commercial tools offering predefined workload scenarios. We exemplarily analyze and optimize our system under test (SUT), which is a rack-based server system from Fujitsu Technology Solutions GmbH. We evaluate only system-compatible components, especially those we can equip in our *PRIMERGY RX200S8* server, such as the processor of the *Intel Xeon E5-26xx* family and 12 different memory modules with various characteristics.

We verify and evaluate the concepts' operating principles that in generally react on heterogeneous workload scenarios, such as customer-specific application software. We define simple component models as separate aspect-based calculation methods, considering the category-specific utilization levels, and show that our abstraction level of the server system as well as its components are sufficiently accurate to calculate the power and temperature. Our simulation model (a hierarchical approach) handles various abstraction levels considering low-level observations, component states or black-box descriptions where we predict the behavior of future components by assuming spreadsheets, for instance. Therefore, we support the scalability and adaptability of the hardware and software characterization, such as diverse component types, quantities, or settings. We analyze the impacts on particular category-based and aspect-based changes considering the adjustments of the technical specification $CH_{TS}$, static characteristic $CH_{CFG}^{ST}$, or dynamic characteristics $CH_{CFG}^{DY}$ in a wide range of heterogeneous workload scenarios. In fact, we identify that the memory-specific read-to-write ratio has a significant impact on the power simulation, which can reduce the over-estimation of the memory power consumption. We observe the highest memory power at the processor-intensive workloads and particularly consider the memory's interaction by the read-to-write ratio in the certain workload scenarios. The mean accuracy of the memory current and obtained power consumptions are in a range of $[+2.1, +14.02]\%$. We specify the processor on a high hierarchy level considering the entire utilization levels and processor characteristics in contrast to the approaches in which the processor is specified by the instruction sets, particular operation traces, or activities on a cycle-by-cycle basis. We linearly define the processor power in relation to the certain p-state (voltage-frequency pair), which becomes non-linear for processors with more than approximately 15 p-states. We demonstrate that the thread-specific utilization levels and frequencies of the processor (*Intel Xeon* architecture) do not improve the accuracy of the processor power consumption. In the case of the processors, we achieve the mean power accuracy of approximately $10\%$, which is sufficiently precise in comparison to related scientific approaches. The power consumption builds the base of the temperature simulation, which we differentiate into a steady and a dynamic phase. We neglect short-term peaks (less than one second) of the power consumption, which have no impact on the memory and processor temperature, because of the thermal inertia. We identify a nearly linear relation between the power consumption and the thermal development, which depends upon the respective component characteristics. In our simulation, we overestimate the memory temperatures in the range between $[\pm 0, +10]\%$, but being more accurate when the memory capacity increases. We demonstrate that our processor temperature calculation is reliable and adequate by the mean accuracy of $[-1.81, +10.44]\%$. The authors of [MAC et al. 2011] state that the components' variability, such as between two processors, is approximately $11\%$, which may affect our accuracy. In general, our evaluations show that our simulation model is sufficiently precise (less than $+10\%$ mean accuracy) when we consider the relevant static and dynamic characteristics of the components at a high level of abstraction. Our accurate and exact component-based models improve the power and thermal calculation of the commercial tools (vendor-based approaches) that consequently avoid over-provisioning when sizing the power supply unit. As a result, we close the gap between the worst-case nameplate values towards more realistic power consumption, as intended in [BBJ et al. 2009]. We simulate authentic workload scenarios instead of worst-case scenarios that reduce the over-estimation of the worst-case power, e.g. the peak power at the full utilization level, and decrease the power gap at lower utilization levels when we compare our simulation results to spreadsheets. We improve the estimation process, especially at lower utilization levels, so that we can define the power consumption in an early design phase more precisely. Additionally, we reduce the measurement effort of actual server systems because we trust in the accuracy and precision of our simulation model, which flexibly react upon the characteristic changes of the components and the particular component series. We consider the vendor experiences, generic findings, spreadsheets, heuristics, and statistics, such as the impacts of the technology designs, generations, or families in which we assume a continuous development of the components that enables our simulation model to predict future and uncertain components. Moreover, we change the component characteristics to support different product life cycle stages and predict the future behavior of the next-generation components.

After verifying and evaluating the basic operating principles on the basis of the isolated components, we simulate the entire server system and analyze the energy efficiency in which we adjust the technical specifications $CH_{TS}$, static characteristics $CH_{CFG}^{ST}$, or dynamic characteristics $CH_{CFG}^{DY}$ as part of the decision variables in our optimization strategy. Therefore, we define a complete server system model that integrates the isolated component models and specifies the communication and interaction between the components that result in the non-linear behavior. Furthermore, we take the thermal control of the complete server system into account. We combine the thermal, power, and performance views of the various components and transform them towards the entire server system. We optimize the server system on the basis of the thermal dissipation and power consumption, as described in Section 5.4.2.1.

We optimize the server system for low utilization because the average server utilization is less than ten percent and always lower than 50 percent in a data center [KFK 2008]. Moreover, a global optimal server configuration saves total costs because of the reduced energy consumption. In our optimization strategy we exemplarily achieve the mean processor power reduction of approximately $53.3\%$. The power optimization has the highest impact at the full utilization levels, but behaves strictly opposite concerning the performance-to-power ratio. Nevertheless, we observe a mean optimization of the performance-to-power ratio by nearly $12.2\%$.

# 8 Conclusion and Future Work

Power and cooling are the key challenges to reducing the greenhouse-gas emissions of server-based computing environments. In this thesis, we present a novel multi-aspect full-system model that simulates and optimizes a wide range of server systems in contrast to traditional full-system simulators. To our knowledge, in academic research, there are no generic approaches that cover the full server system simulation on a common base. In our proposed prototypical implementation in MATLAB/Simulink, we explicitly cover the heterogeneous characteristics of the hardware and software variations.

## 8.1 Summary

We develop a hierarchical and abstract approach that provides the opportunity to define the system from upper to lower abstraction levels. We specify a generic, flexible, and scalable configuration tree as a static part of our concept that defines the actual physical customer-specific system configuration from the structural perspective. Herein, we define the encapsulated layers: *configuration*, *logical and physical,* and *process and control*, which we define separately from each other, allowing to support independent descriptions of the diverse domain-specific characteristics. We abstract the server system complexity and include the configuration adaptability to support architectural and structural changes at the physical domain. Our decoupled and hierarchical concept provides the availability to add new components at various abstraction levels.

In the *configuration* layer, we specify the architecture, design, and structure of the entire server system to support multiple server generations. We do not merely consider the actual configuration, but also take the maximal amount of possible mountable system-board components into account. We specify the five major categories – *processor*, *memory*, *input/output*, *fan*, and *others* – in the generic, but static configuration tree. Moreover, we model and characterize each component of the server system primarily in the physical domain, using a mix of commercial and academic algorithms in the *configuration* layer.

We define the mathematical methods to calculate the multi-aspects of each component considering the static configuration tree, which we further define by their technical specification and their respective characteristics. Accordingly, the *logical and physical* layer builds the base of our simulation-based model, in which we specify the component-based power, thermal, and performance models as a set of utilization-based functions describing the non-linear behavior. We propose a flexible category-specific classification and generic characterization approach to support their corresponding calculation-based methods. In general, we consider the relevant aspect-based characteristics of the components concerning their explicit category within the configuration tree and apply a weight coefficient to distinguish their particular significance of the certain aspects.

In Section 7.2, we demonstrate the precise accuracy of our power and thermal calculation concerning the memory modules and processors. In our approach, we group the memory states and abstract the explicit memory accesses to reduce complexity. We exclusively consider the utilization levels instead and found that a precise simulation requires additional data about the memory instructions because the measured power values differ in relation to the benchmark when tracing the same utilization level. As a result, we define the read-to-write ratio of the memory modules to compensate concrete accesses and always estimate them in the category-bounded workload scenario based on empirical studies by a statistical approximation. The precise power consumption builds the base of the temperature simulation, which we differentiate into a steady state, an increasing, and a decreasing temperature development. We neglect short-term peaks (less than one second) of the power consumption, which have no impact on the memory and processor temperature because of the thermal inertia. We demonstrate that our simple state-based memory model sufficiently accurately calculates the power and temperature, which can reduce the over-estimation. We do not require fine granular low-level data, such as instructions, which reduces the estimation effort of the memory modules. Moreover, we demonstrate sufficiently precise results of our decoupled processor models concerning the *Intel Xeon* architecture, in which we neglect the thread-specific utilization levels and frequencies. We precisely simulate the processor power and temperature, which are only based upon the utilization levels. In our processor models, we consider the mean usage of the integrated subsystems in which we neglect the specific operations and the fan-based cooling effect. Our accurate and exact power calculation reduces the over-provisioning of the server system, particularly in industrial practice and, in consequence, optimizes the PSU (power supply unit) sizing.

In Section 7.3, we demonstrate that a sufficiently precise power simulation requires static as well as dynamic characteristics. We present that our approach is sufficiently scalable and sensitive about varying the compatible subset of the possible components concerning the category-based and aspect-based characterization to cover a variety of server systems. We concentrate upon the power-specific calculation methods because the power consumptions of the various components are accessible everywhere and available to anyone. In contrast to common assumptions, we found the following relevant characteristics of the memory-based calculation methods: *vendor*, *die*, *series*, *fabrication size*, *synchronization mode*, and *ranks*. We consider the single characteristic changes in our power calculation, but in the majority of the cases, the characteristics influence each other and cannot be calculated separately. Thus, we define a system of linear equations in a matrix to specify the interdependencies of the technology-based characteristics, such as *capacity*, *density*, *rank*, and *rank linking*. We simulate equivalent results concerning precision and accuracy in comparison to academic approaches. In addition, our simulation-based results of the *E5-2600v2* product family are reliable and adequate. We observe the processor-specific characteristics: *semiconductor technology* (*thermal design power*), *product life cycle stage*, *fabrication size*, and *series*, which have a significant impact on our calculation-based methods. In our approach, the processor power is linear to the certain p-state (voltage-frequency pair), and we assume that the processor frequencies rely upon the utilization level. We define a non-linear power method when the amount of processor-specific p-states is bigger than 15. In fact, the amount of processor cores and the cache sizes are additional relevant characteristics of processor power calculation.

Our simulation-based model flexibly reacts upon the characteristic changes of the components, which finally reduces the measurement effort of the particular component series. We demonstrate that we can forecast future generations of high-performance systems and components by assuming the predecessor or a similar generation that follows the same trend as the components before. We demonstrate that our model is universally applicable on next-generation and uncertain components. Our concept supports the virtual prototype of a component or server system, which is a benefit in the early design stage and a unique selling proposition of our thesis.

After separately defining each calculation-based method and their corresponding characteristics, we specify the relations between the aspects of a single component as a dynamic behavioral description characterized by findings of a real-life system and its respective hardware. Furthermore, we define the component-specific relations of the entire system behavior in MATLAB, because a component can influence the behavior of another component. We consider the interdependencies between the components, which results in more realistic power and temperature calculation. The *process and control* layer includes the dynamic system behavior, such as the inter- and intra-component communication (interactions) between the components, which we implement in Simulink considering the connectors and the architectural description of the *configuration* layer. The description of the dynamic behavior enables restriction on the actual performance when the resources are limited or we exceed the critical temperatures.

Our generic simulation-based model is based upon the operational models described in the encapsulated *configuration*, *logical and physical,* and *process and control* layers. Herein, we support the variance of server systems and components because of the flexible category-specific configuration and characterization that we consider as a centralized database. We provide access to individually server system configuration within the database to enable the use of our models across multiple families and generations. Our simulation-based model offers the opportunity to use white-box, gray-box, and black-box approaches from upper to lower abstraction levels, because of the component-based encapsulation and different levels of abstractions in the hierarchical configuration tree. In contrast to academic approaches, we concentrate upon an exact server system configuration within one simulation run, but do not dedicate the model to an explicit server system or particular workload scenario. We describe the external environment and constraints, such as the ambient temperature or thermal limits, and integrate the hardware-based offline settings, such as the BIOS/UEFI configuration, which we consider in our calculation methods. We abstract the software dependencies, such as the specific operating system, but consider the corresponding weight coefficients instead. The customer may vary the external and internal data from a lower level to an upper level of the abstracted server system to specify the component and system behavior. This avoids extra measurements for a new configuration or other environmental conditions.

We abstract low-level data, such as instructions, and ignore the particular hardware-specific events, e.g. performance counters that certainly rely upon the exact architecture. Beneficially, our model is independent of the architecture and generation; because of the abstraction, we reduce the model complexity. For the purposes of the multi-aspect server simulation, we transform the steady workload to continuous values to optimize the energy efficiency in the long-term. Our flexible concept allows the definition of the customer-specific and realistic workload scenarios based upon the category-specific utilization levels, which specifies the time-continuous and value-continuous stimuli for the simulation. We predefine particular utilization levels in various workload scenarios to be compatible with the commercial tools that under-utilizes all components or define a worst-case workload scenario that fully utilizes all categories, for instance. Herein, we differentiate into category-bounded (processor-bounded, memory-bounded, or input/output-bounded) workload scenario that preliminarily sets the focus on what to concentrate on the server system optimization.

We distinguish our optimization strategy into the cascading primary and secondary phases, which we differentiate into the short-term and long-term strategies. We specify the primary phase as online because we consider the short-term modifications when the server system is working, and we guarantee that our variations do not have any negative impact on the system or component’s performance. We adjust the dynamic characteristics considering the thermal-based and power-based management techniques. We consider the common approaches, such as the dynamic voltage frequency scaling (DVFS) and the dynamic thermal management (DTM), to represent the short-term behavior (dynamic characteristics). We specify the cascading primary (online) and secondary (offline) phases to alter the relevant characteristics and configurations. We control the internal system temperature considering the ambient temperature in the dynamic thermal management and vary the dynamic component characteristics considering the short-term strategies. The abstract configuration changes of the static characteristics and technical specification is our secondary optimization strategy. When the server system executes a workload for hours or days, the short-term management techniques are insignificant. As a contrast to the online phase, we specify the secondary phase, which is an offline optimization, whereby the changes have indirect influences on the primary phase in which we adjust the static characteristics or technical specifications. We concentrate upon the power optimization considering uncommon-case working conditions, such as low-intensive utilization levels, which academic approaches neglect. The thermal control autonomously reacts upon an increasing temperature, which exceeds a predefined threshold and raises the cooling airflow according to the fully functional level of the server system. We control the system and component temperatures within the reliability and functional level by enabling the dynamic thermal management techniques. We reduce the average and peak power of the components by their dynamic characteristics under the condition that the performance is not affected as autonomously done by every ACPI-based operating system, which optimize the worst-case power consumption in a time horizon of seconds or minutes. We consider the global system thermal management and the local power management technique at the same time, whereby the fraction of the fan power is less in comparison to the component power. We separately optimize each component, but are aware of the negative impacts on the global system. The most adjustments in the secondary phase are applicable when the system is off and we concentrate upon the long-term strategies to optimize the server system in the early design phase, for instance. We disable the hardware-based features that do not result in performance loss, such as the processor virtualization before the system starts. We adjust the BIOS/UEFI characteristics that are significant to the server system energy efficiency ratio, but constant over hours or days, particularly in a data center. We support the flexible adjustment of the hardware configurations, especially their static characteristics.

In our concept, we optimize a server system at each time step when the utilization levels change. We analyze the actual results and decide on the adequate management technique in the optimization strategy, according to the two optimization phases. We analyze the impact of our changes concerning the energy-to-performance ratio in comparison to the base energy efficiency of the initial server configuration. The adjustments of the server configuration and characterization require an additional calculation, which results in an iterative approach.

We alter the decision variables on the basis of the cascading phases of the optimization strategy to provide the most probable presentation of the running server system, while adjusting all dynamic characteristics. We optimize the server system in the secondary phase, assuming the changes in the primary phase, wherein we modify the static characteristics, which require a repetition of the adjustments concerning the dynamic characteristics. We have to consider any characteristics and configurations in the entire design space, whereby we explore single decision variables or multiple combinations. We optimize the technical specification, which requires the alternation of the dynamic and static characteristics. In principle, we consider the hierarchical order of our configuration tree and use a bottom-up strategy to alter the decision variables beginning with the dynamic characteristics up to the static configuration. We restrict the design space of each category on the basis of the impacts on the dynamic characteristics, static characteristics, or technical specification. Accordingly, we dynamically annotate the classes and characteristics of the tree in relation to the actual management technique on the basis of the aspect. The abort criteria restrict the design space, and in consequence, we do not consider all characteristics in the configuration tree. We compare the energy efficiency ratios and assume that the usefulness of our adjustments are represented by a scalar value of each aspect and finally in the energy efficiency ratio.

The multi-objective optimization provides a ranked list in descending order that specifies the set of optimal solutions when the present energy efficiency is better than the base energy efficiency. We select the globally optimized server configuration and characteristic that dominates for the longest period of time; but to be more precise, we optimize the balance of the power and respective energy in the integrand of the minimized time integral.

In Section 7.4, we demonstrate the possible improvements that result upon the server system optimization. Moreover, the simulation optimizes the energy efficiency of the server system at various utilization levels, especially at low-intensity phases (under-utilization). We demonstrate that we improve the energy efficiency when we optimize the components concerning the specific demand and avoid under-utilized components to improve the non-peak efficiency considering the low-intensive workloads.

Our flexible simulation model can reduce the measurement effort because the model may vary the server configuration in a short time and can simulate the entire system, including a wide range of spreadsheet data, observations, statistical results, or customer-specific intellectual properties.

We present the plausibility of our component-based models on the basis of the sufficiently accurate power and thermal results. We demonstrate that our abstract simulation model provides the entire system complexity and at the same time is simple enough to cover the system behavior considering the most important characteristics and configuration.

We demonstrate the feasibility and advantages of our concept through our prototype implementation, in which we empirically validate our server system using a variety of artificial workloads to ensure the reproducibility at any time. We address the significant static as well as dynamic characteristics and configurations of the precise calculation of the aspects.

Our simulation results are more accurate in comparison to the results gained from commercial tools. We reduce the over-estimation of the worst-case power, e.g. the peak power at the full utilization level, and decrease the power gap at lower utilization levels when we compare our simulation results to spreadsheets, and we can easily reduce the power gap of the measurement-based approaches. We improve the estimating process so that we can define the power consumption in an early design phase more precisely, which may improve the power estimation in industrial practice.

## 8.2 Future Work

We present our multi-aspect full-system server model and optimization concept as a simulation-based approach. However, we make several assumptions and limitations in this thesis. In the following section, we present a couple of possible improvements concerning our proposed simulation-based model and discuss some recommendations for future research.

The first limitation in our simulation model is that we neglect the component thermal control, either as an automated mechanism to cool down the explicit device internally or as a server-specific technology to cool down the components externally. The fan settings are vendor-specific and are impossible to be changed for a customer-specific thermal control. The customer cannot directly influence the server's noise, airflow, or temperature behavior. Improvements could be made to the temperature accuracy of the entire system when we consider the intra- and inter-component airflow and observe the mutual influence of the heat dissipation. Additional studies could assess the integration of the computational fluid dynamics within a server system to plan the ventilation and define the critical thresholds as part of the thermal control. In our prototype implementation, we consider the linear fan algorithm that does not require a continuous control process to a target temperature. Future research could also concentrate on the extension of the thermal control, such as studying a closed-loop (proportional integral derivative – PID) mechanism that might save additional energy. The power and thermal management techniques may act differently in certain circumstances in comparison to the common usage and behavior, which result in future research.

Furthermore, the second limitation is that we restrict the hardware configuration towards the rack-based and tower-based server systems because we want to reduce the degree of freedom concerning our simulation model. An improvement could be made to support a blade chassis, because blade servers have been ignored so far. In any case, supporting a blade server will increase the adjustment effort to model a server system because we have to consider the specific slots to mount several devices of the prewired chassis and shared components. The actual model assumes only the use of system-board components. Therefore, further study could focus on the optional (add-in) and on-board components. Another limitation of our simulation-based approach is that we define the input/output and other resources as static models assuming empirical measurements. Additional improvements could be made to the simulation when modeling the dynamic behavior of the *input/output* and *others* components. Our simulation model is actually limited to the server-specific generations of the *Intel Xeon* processors, especially *Sandy Bridge*, *Ivy Bridge*, and *Haswell*, because the experimental results of Fujitsu are not accessible to the public and are restricted to these generations. In particular, we concentrate on the processors of the *E5-2600v2* product family. An extra study could evaluate other processor generations and series.

In general, we need to extend the component models for the next generations and require future research that concentrates on updating the dependencies and related weight coefficients. We recommend further investigations on next-generation components and server systems to specify the generation-specific static and dynamic characteristics.

Another assumption is that we consider a continuous development of the system and components, which may cause wrong expectations of the applicability of our model concerning next-generations. In our approach, we consider the vendor experiences, spreadsheets, heuristics, and statistics to estimate the future systems, assuming the earlier observations at every product phase. The qualities of the generic findings, e.g. the defined weight coefficients based on the real-life measurements, highly influence the calculation methods. We assume that our weight coefficients can become more accurate, which requires further studies.

Another assumption is related to the system model because we manually predefine the server system architecture, communication, and connectors. Additional improvements could be made to the automated definition on the basis of the system specification. Moreover, we assume that the updated behavioral description that relies on the static and dynamic characteristics has a high potential for improvements. Another study could investigate on dynamically adjusting the settings of the class-specific characteristics of our configuration tree during the simulation. The actual hierarchical model abstracts the irrelevant features, e.g. the features that are specific to a single server system, because of their little effect on energy efficiency. Further improvements could be made to the accuracy of our calculation-based methods or possible characteristic changes in our optimization strategy. We assume that creating rough models considering the technical specification can be as adequate as highly detailed models, which, in consequence, creates less computational effort.

Further research could also concentrate on a more granular component model, which might have a positive impact on the accuracy of our power and temperature calculation. In our simulation model, we restrict the performance models by considering the performance scores that rely upon the real-life measurements. We need further investigation on the performance scores concerning the impact of category-based and aspect-based changes of the characteristics and configurations to calculate the performance explicitly.

This thesis presents a variety of artificial workload scenarios to ensure the reproducibility at any time. We evaluate benchmarks considering synthetic workloads to test real-world systems or discrete system components in a specified and repeatable manner under defined circumstances. Future research could also concentrate on evaluating customer-specific, real-life utilization traces and input/output-bounded workload scenarios. Additional improvements could be made to the dynamical changing of the component-specific utilization levels at simulation time, defined by the customer. Another limitation is the actual restriction of the utilization-based scenarios and models. A further study assesses the extension of our workload scenarios concerning the explicit low-level data. Thus, we need detailed design data, such as the architecture, structure, transitions, execution units, registers, or an activity at the circuit level in the physical system domain to define the physical and logical models as a white-box approach that suitably reacts to the detailed low-level data and considers the internal data flow. We assume that the extra information increases the model complexity that further result in higher simulation time. In general, the explicit workload scenario is a continuous stimulus of our simulation-based model, in which we calculate the corresponding power independently of the operating system. A further study could explore the usefulness of including the OS-specific timings or scheduling.

In the interests of simplicity, we follow a purely greedy approach in our optimization strategy, which is a further limitation. We may escape from local minima of the greedy approach when we use a metaheuristic algorithm. In our simulation framework to reduce the risk of a local minimum, we do not specify an explicit algorithm. Future research could concentrate on analyzing the acceptable probability to seek a global optimum and specifying the best iterative approach, such as Kernighan-Lin, Simulated Annealing (SA), Evolutionary Algorithms (EA), or Genetic Algorithms (GA). We could not integrate the cascading phases of the short-term and long-term optimization into a real server system, because we have to disable a couple of features and change the firmware, which may result in an unstable server system. The internal system sensors limit the execution of our concept because of their latency and bus bandwidth. In addition, the embedded controller that stores the firmware does not provide sufficient performance and storage capacity to execute our algorithm. Future research could assess the possibility of integrating our algorithm. Another limitation of our optimization is that we restrict the alternation of all characteristics and configurations in the entire design space, exploring only single decision variables or certain combinations, because the process is too expensive in terms of simulation time and performance requirements. Hence, we do not completely traverse the configuration tree considering all possible characteristics and configurations. Additional improvements could be made to the selection of adequate heuristics in the alternation strategy, which avoids the disproportionate increase in the alternation complexity (design space reduction) and provides flexible abort criteria to reduce the corresponding simulation time. We specify the knowledge-based and vendor-specific alternation rules that exclude irrelevant adjustments, such as insignificant modifications, and define a preference of the decision variables to reduce the optimization effort. We assume further research on adequately adjusting the alternation rules of next-generation systems and components.

Another limitation is that we implement the post-process as an offline optimization and decide on a global optimal solution after simulating the server system, because we cannot adequately store the set of local optimal solutions in Simulink. Therefore, we save the temporary results during the simulation and analyze them afterwards. How to solve the problem is a further analysis question.

# A1. Nomenclature

Table 86: Nomenclature A1.1

| Nomenclature | Meaning |
|---|---|
| $A, A_j$ | Aspect $j$ |
| $C, C_i$ | Component $i$ |
| $CS, cs_i$ | System-board category $i$ ( $\equiv$ components) |
| $CL, CL_i$ | Classes $i$ |
| $CH, CH_i$ | Characteristic $i$ |
| $EX$ | Externals |
| $SY$ | System characterization, model |
| $FS$ | Full-system simulation and optimization |
| $M_{SY}^{EX}$ | Mapping between externals and system |
| $SW$ | Application software |
| $SH$ | Hardware |
| $HC$ | Components |
| $CA$ | Add-in components |
| $CO$ | On-board components |
| $HO$ | Connectors |
| $HP$ | Power supply |
| $IN$ | Input |
| $PA$ | Parameter |
| $OUT$ | Output |
| $ST_s$ | Static characteristics |
| $DY_s$ | Dynamic behavior of the system (characteristics) |
| $EE, EE_{BASE}$ | Energy efficiency |
| $PO$ | Power |
| $PE$ | Performance |

| Nomenclature | Meaning |
|---|---|
| $i, j, n, m, k$ | Index |
| $N_0$ | Any natural number $N_0 = \{0,1,2,3,\ldots\}$ |
| $\mathbb{R}$ | Any positive real number |
| $\alpha, \hat{\alpha}$ | Environment conditions |
| $\beta, \hat{\beta}$ | Characteristics |
| $\gamma, \hat{\gamma}$ | Management techniques |
| $\delta, \hat{\delta}$ | Communication |
| $\varepsilon$ | Simulation results |
| $\eta$ | External constraints |
| $\upsilon$ | Internal constraints |
| $\chi$ | Simulation constraints |
| $\xi$ | Software settings |
| $\sigma$ | Abort criterion |
| $v_n,$ | Vertex $n$ |
| $v_0$ | Root vertex |
| $e$ | Edge |
| $proc$ | $Processor$ |
| $mem$ | $Memory$ |
| $io$ | $Input/output$ |
| $oth$ | $Others$ |
| $L_1$ | First level of a certain tree |
| $P_{id}$ | Power consumption of $id$ *id ={processor, OS, dynamic, static, state, cache, transition, core, leakage, server}* |
| $t_{ij}$ | Transition from $i$ to $j$ |
| $S_i$ | State $i$ |
| $K_i$ | Coefficient $i$ |

Table 87: Nomenclature A1.2

| Nomenclature | Meaning |
|---|---|
| $\boldsymbol{TH}$ | Thermal |
| $\boldsymbol{AC}$ | Architecture |
| $\boldsymbol{CC}$ | Connectors |
| $\boldsymbol{MAS_C}$ | Aspect-based models per component |
| $\boldsymbol{A_{j_{C_i}}}$ | Element in matrix $MAS_C$ |
| $\boldsymbol{F_{A_{j_{C_i}}}}$ | Functional description of $A_{j_{C_i}}$ |
| $\boldsymbol{F(x)}$ | Objective functions with decision variables x |
| $\boldsymbol{x}$, $\boldsymbol{\breve{x}}$ | Decision variables |
| $\boldsymbol{G(x)}$, $\boldsymbol{G_1^C(x)}$ | Constraints of $F(x)$ |

| Nomenclature | Meaning |
|---|---|
| $\boldsymbol{f_{min}, f_{max}}$ | Minimal and maximal frequency |
| $\boldsymbol{T_{case}}$ | Processor case temperature |
| $\boldsymbol{w_{il}}$ $\boldsymbol{W, W_{T_u}}$ | Workload for component $m$ at a time $t_k$ |
| $\boldsymbol{WP_k}$ | Workload profile $k$ |
| $\boldsymbol{T}$ $\boldsymbol{T_s, T_u, T_w}$ | Time period simulation, utilization, workload |
| $\boldsymbol{t_k}$ | Time step $k$ |
| $\boldsymbol{\Delta T_F}$ | Time amplifier for workload |
| $\boldsymbol{\Delta T, \Delta T'}$ | Time resolution |
| $\boldsymbol{P_m, P_i, P_c, P_u}$ | Performance counters of memory bus transactions, instructions, clock cycles, last-level cache references |

Table 88: Nomenclature A1.3

| Nomenclature | Meaning |
|---|---|
| $AX, AY$ | Aspect-based models |
| $R_A, R_{A_C}$ | Aspect-based relations |
| ${}_{A_j}R_{t_k}^{t_{k+1}}$ | Impact of $A_j$ at time step $t_{k+1}$ |
| $BE_C, BE_{C_i}^{A_j}$ | Component behavioral model |
| $R_{A_l}^{A_k}$ | Relation between the aspects $(A_k, A_l)$ |
| $R_{A_{jC_i}}$ | Aspect-related relevance for component i and aspect j |
| $RR_{A_{j_{rel}}}^{A_{k_{rel}}}$ ${}_{min}RR_{A_{j_{rel}}}^{A_{k_{rel}}}$ ${}_{max}RR_{A_{j_{rel}}}^{A_{k_{rel}}}$ | Interval limits of relation $R_{A_l}^{A_k}$ |

| Nomenclature | Meaning |
|---|---|
| $\vec{u}_m = u_{CS_i}$ $\vec{u}_{t_k}$ | Utilization level of component category $m$ time step $t_k$ |
| $\theta$ $\theta_R, \theta_C, \theta_S$ $\theta_{TS}, \theta_{CS}$ | Configuration tree (HW, SW) released, customer, system-compatible technical specification, characteristics |
| $\theta_C^l,\ {}_k\theta_C^l$ $\theta_C^l(x)$ | Customer-specific configuration at iteration $l$, time step $k$, and decision variable $x$ |
| $WF_{A_{jC_i}}^{CH_k}$ | Weighting factor component $i$, aspect $j$, and their characteristics $CH_k$ |
| $R_{BE}$ | Relations between the component-specific behavior models |
| $CH, CH_{TS}$ $CH_{CFG}^{ST}, CH_{CFG}^{DY}$ | Characteristics: technical specification, static, dynamic configuration |
| $WF_{A_{jC_i}}^{CH}$, $WF_{A_{jC_i}}^{CL_{TS}}$, $WF_{CFG}^{ST}$, $WF_{CFG}^{DY}$ | Weight coefficients: component $i$, aspect $j$, and their characteristics, class, static, dynamic |

## A2. List of Abbreviations

Table 89: List of abbreviations A2.1 – (A - CPRH)

| Abbreviation Acronyms | Definition |
|---|---|
| A | Ampere, Amps |
| AC | Alternating Current |
| ACPI | Advanced Configuration and Power Interface |
| ADD | Integer addition |
| AE | Airflow Efficiency |
| AEU | Air Economizer Utilization |
| AFC | Active Flow Control |
| ALU | Arithmetic Logic Unit |
| AND | And |
| API | Application Programming Interfaces |
| APM | Advanced Power Management |
| ASHRAE | American Society of Heating, Refrigerating and Air-Conditioning Engineers |
| ASIC | Application-Specific Integrated Circuit |
| ASP | Application Service Provider |
| AWS | Amazon Web Services |
| B2C | Business to Consumer |
| BFS | Breadth-First Search |
| BIOS | Basic Input Output System |
| BMC | Baseboard Management Controller |
| BOM | Bill Of Material |
| BRR | Bank Round Robin |
| BX | Blade server |
| CADE | Corporate Average Datacenter Efficiency |
| Capex | Capital Expenditures |
| CCG | Cache Conflict Graph |
| CD | Compact Disc |
| CEEDA | Certified Energy Efficient Data Center Award |
| CEF | Carbon Emission Factor |
| CFD | Computational Fluid Dynamics |
| CFG | Control Flow Graph |
| CI | Capture Index |
| CKEH | Clock Enable High |
| CKEL | Clock Enable Low |
| CMOS | Complementary Metal-Oxide Semiconductor |
| COP | Coefficient Of Performance |
| CPI | Cycles Per Instruction |
| CPRH | Command Pair Rank Hopping |

Table 90: List of abbreviations A2.2 – (CPU - EPS)

| Abbreviation Acronyms | Definition |
|---|---|
| CPU | Central Processing Unit |
| CRAC | Computer Room Air Conditioner |
| CRAH | Computer Room Air Handler |
| CSE | Data Center Cooling System Efficiency |
| CSS | Cooling System Sizing |
| CUE | Carbon Usage Effectiveness |
| CUT | Circuit Under Test |
| DB2 | Database |
| DC | Data Center |
| DC | Direct Current |
| DC-DC | Direct Current to Direct Current converter |
| DCeP | Data Center energy Productivity |
| DCiE | Data Center infrastructure Efficiency |
| DDR | Double Data Rate Synchronous Dynamic Random-Access Memory |
| DES | Discrete Event Systems |
| DESS | Differential Equation Specified Systems |
| DFA | Deterministic Finite Automaton |
| DFD | Data Flow Diagram |
| DFS | Depth-First Search |
| Die | silicon device (chip) |
| DIMM | Dual In-line Memory Module |
| DOM | Date Of Manufacture |
| DPC | DIMMs Per Channel |
| DPM | Dynamic Power Management |
| DRAM | Dynamic Random-Access Memory |
| DS | Data Sheet |
| DSP | Digital Signal Processor |
| DTM | Dynamic Thermal Management |
| DTS | Digital Thermal Sensor |
| DTSS | Discrete Time Specified Systems |
| DVFS | Dynamic Voltage Frequency Scaling |
| DWPE | Data Center Workload Power Efficiency |
| EA | Evolutionary Algorithms |
| EC2 | Amazon Elastic Compute Cloud |
| EIS | Enterprise Information System |
| EIST | Enhanced Intel SpeedStep Technology |
| EOL | End Of Life |
| EoR | End of Row |
| EOS | End Of Sale |
| EPA | US Environmental Protection Agency |
| EPS | Energy Power Supply |

Table 91: List of abbreviations A2.3 – (ERD - ILPA)

| Abbreviation Acronyms | Definition |
|---|---|
| ERD | Entity-Relationship Diagrams |
| ERF | Energy Reuse Factor |
| ESL | Electronic System Level |
| ESM | Energy State Machine |
| ESSA | Dell Energy Smart Solution Advisor |
| FAST | FPGA-Accelerated Simulation Technologies |
| FB-DIMM | Fully-Buffered Dual In-line Memory Module |
| FCFS | First Come First Serve |
| FIFO | First In First Out |
| Flop/s FLOPS | Floating-Point Operations Per Second |
| FM | Functional Model |
| FPGA | Field Programmable Gate Array |
| FPU | Floating-Point Unit |
| FSB | Front Side Bus |
| FSC | Fan Speed Control |
| FSM | Finite State Machine |
| FSP | Full Service Provider |
| FTA | Fault Tree Analysis |
| GA | Genetic Algorithms |
| GCPI | Green Computing Performance Index |
| GEC | Green Energy Coefficient |
| GEMS | General Execution-driven Multiprocessor Simulator |
| GFLOPS / TFLOPS | Giga / Tera Floating-Point Operations Per Second |
| GPU | Graphics Processing Unit |
| GUI | Graphical User Interface |
| HDD | Hard Disk Drive |
| HPC | High Performance Computer |
| HVAC | Heating Ventilation Air Conditioning |
| HVACR | Heating Ventilation Air Conditioning And Refrigeration Technology |
| HW | Hardware |
| I/O | Input / Output |
| I2C, I$^2$C | Inter-Integrated Circuit Bus |
| IaaS | Infrastructure as a Service |
| ICMB | Intelligent Chassis Management Bus |
| ICT | Information and Communication Technologies |
| IDC | International Data Corporation |
| IDD | Drain current of a CMOS circuit |
| IEC | International Electrotechnical Commission |
| ILPA | Instruction-Level Power Analysis |

Table 92: List of abbreviations A2.4 – (IMC - MOV)

| Abbreviation Acronyms | Definition |
|---|---|
| IMC | Integrated Memory Controller |
| INSEE | Interconnection Network Simulation and Evaluation Environment |
| IP | Intellectual Property |
| IPC | Instructions executed per Cycle |
| IPMB | Intelligent Platform Management Bus |
| IPMI | Intelligent Platform Management Interface |
| IPTV | Internet Protocol Television |
| IQR | Interquartile Range |
| iRMC | Integrated Remote Management Controller |
| ISA | Instruction Set Architecture |
| ISO | International Organization for Standardization |
| ISP | Internet Service Provider |
| ITEE | IT Equipment Efficiency |
| ITEU | IT Equipment Utilization |
| J | Joule |
| JEDEC | Joint Electron Device Engineering Council |
| JRE | Java Runtime Environment |
| JVM | Java Virtual Machine |
| KPI | Key Performance Indicator |
| KVM | Kernel-based Virtual Machine |
| kW, kWh | Kilowatt, Kilowatt hour |
| L2, L3 | Level two / three cache |
| LC | Liquid Cooling |
| LCM | Least Common Multiple |
| LLC | Last-Level Cache |
| LOC | Lines of Code |
| LTS | Long-Term Support |
| LTV | Linear Time-Varying |
| LUT | Lookup Tables |
| LV | Low Voltage |
| LXC | Linux Container Virtualization |
| MBD | Model-Based Design (Development) |
| Mbps | MegaBits Per Second |
| MBSE | Model-Based Systems Engineering |
| MFSMOS | Multi-aspect Full-system Server Model and Optimization Concept as a Simulation-based Approach |
| MIPS | Millions of Instructions Per Second |
| MIPS | Microprocessor without Interlocked Pipeline Stages Architecture |
| MO, MOO | Multi-Objective Optimization |
| MOP | Multi-Objective Optimization Problem |
| MOV | Load and store control registers |

Table 93: List of abbreviations A2.5 – (MPEG - PROCHOT)

| Abbreviation Acronyms | Definition |
|---|---|
| **MPEG** | **M**oving **P**icture **E**xperts **G**roup |
| **MTBF** | **M**ean**t**ime **B**etween **F**ailure |
| **MTTF** | **M**ean**t**ime **T**o **F**ailure |
| **MVC** | **M**odel-**V**iew-**C**ontroller |
| **MW** | **M**ega**w**att |
| **MySQL** | **My** **S**tructured **Q**uery **L**anguage |
| **NAS** | **N**etwork-**A**ttached-**S**torage |
| **NBTI** | **N**egative **B**ias **T**emperature **I**nstability |
| **NDF** | **N**umerical **D**ifferentiation **F**ormulas |
| **NFA** | **N**ondeterministic **F**inite **A**utomaton |
| **NUMA** | **N**on-**u**niform **M**emory **A**ccess |
| **O&M** | **O**peration **A**nd **M**aintenance |
| **ODE** | **O**rdinary **D**ifferential **E**quations |
| **OMAP** | **O**pen **M**ultimedia **A**pplication **P**latform |
| **OMT** | **O**bject-oriented **M**odeling **T**echniques |
| **Opex** | **Op**erational **Ex**penditures |
| **OPS** | **O**perations **P**er **S**econd |
| **OR** | **Or** |
| **OS** | **O**perating **S**ystem |
| **PaaS** | **P**latform **as a** **S**ervice |
| **PC** | **P**erformance **C**ounter |
| **PC²** | **P**aderborn **C**enter for Parallel **C**omputing |
| **PCB** | **P**rinted **C**ircuit **B**oard |
| **PCI** | **P**eripheral **C**omponent **I**nterconnect |
| **PCIe** | **P**eripheral **C**omponent **I**nterconnect **E**xpress |
| **PDLC** | **P**roduct **D**evelopment **L**ife **C**ycle |
| **PDU** | **P**ower **D**istribution **U**nit |
| **PF** | **P**ower **F**actor |
| **PFC** | **P**ower **F**actor **C**orrection |
| **PFLOPS** | **P**eta **F**loating-Point **O**perations **P**er **S**econd |
| **PHY** | **PHY**sical layer transceiver, Ethernet chip |
| **PID** | **P**roportional **I**ntegral **D**erivative |
| **PLA** | **P**ost Layout **A**nalysis |
| **PLC** | **P**roduct **L**ife **C**ycle |
| **PMBus** | **P**ower **M**anagement **Bus** |
| **PMC** | **P**erformance **M**onitoring **C**ounters |
| **PO** | **P**areto **O**ptimal |
| **POST** | **P**ower-**O**n **S**elf-**T**est |
| **PPC** | **P**RIMERGY **P**ower **C**alculator |
| **PPW** | **P**erformance **p**er **W**att |
| **PROCHOT** | **Proc**essor **Hot** |

Table 94: List of abbreviations A2.6 – (PSG - SMI)

| Abbreviation Acronyms | Definition |
|---|---|
| PSG | Product Sales Group |
| PSM | Power State Machine |
| PSU | Power Supply Unit |
| PTU | Intel Power Thermal Utility |
| PUE | Power Usage Effectiveness |
| PVT | Process, Voltage, and Temperature |
| PWM | Pulse Width Modulation |
| QPI | Quick Path Interconnect |
| RAID | Redundant Array of Independent Disks |
| RAM | Random-Access-Memory |
| RAMS | Reliability, Availability, Maintainability, and Safety |
| RC | Thermal Resistances and Capacitances |
| RCI | Rack Cooling Index |
| RDIMM | Registered Dual In-line Memory Module |
| RDP | Remote Desktop Protocol |
| RDU | Rack Distribution Unit |
| RHEL | Red Hat Enterprise Linux |
| RHI | Return Heat Index |
| RIFF | Read or Instruction Fetch First |
| RMS | Root-Mean-Square |
| ROI | Return On Investment |
| RPM | Revolutions per Minute |
| RTI | Return Temperature Index |
| RTL | Register-transfer Level |
| RX | Rack server |
| SA | Switching Activity |
| SA | Simulated Annealing |
| SA/SD | Structured Analysis / Structured Design |
| SaaS | Software as a Service |
| SAN | Storage-Area-Network |
| SCI | System Control Interrupt |
| SDA | Serial Data Input/Output |
| SDR | Sensor Data Record |
| SDRAM | Synchronous Dynamic Random-Access Memory |
| SERT | Server Efficiency Rating Tool |
| sf | Square Feet |
| SHI | Supply Heat Index |
| SLES | SUSE Linux Enterprise Server |
| SMBus | System Management Bus |
| SMD | Surface-Mounted Device |
| SMI | System Management Interrupt (OS) |

Table 95: List of abbreviations A2.7 – (SMI - VRM)

| Abbreviation Acronyms | Definition |
|---|---|
| SMI | Scalable Memory Interface (processor) |
| SMT | Simultaneous Multithreading Technology |
| SoC | System on a Chip |
| SPEC | Standards Performance Evaluation Corporation |
| SQL | Structured Query Language |
| SRAM | Static Random-Access Memory |
| SSD | Solid-State Drives |
| SSJ | Server-side Java |
| STD | State Transition Diagram |
| STG | State Transition Graph |
| SUB | Subtract |
| SUE | Server Utilization Effectiveness |
| SUT | System Under Test |
| SVN | Subversion |
| SW | Software |
| SWaP | Space, Watts and Performance |
| TA | Transient Analysis |
| TCO | Total Cost of Ownership |
| TDDB | Time-Dependent Dielectric Breakdown |
| TDP | Thermal Design Power |
| TGG | The Green Grid |
| THD | Through-Hole Device |
| THERMTRIP | Thermal Trip |
| tick-tock | Intel specific model for technology cycles |
| TIV | Time Invariant |
| TM | Timing Model |
| ToR | Top of Rack |
| TPC | Transaction Processing Performance Council |
| TX | Tower server |
| U | Rack Unit |
| U.S. / US | United States |
| UEFI | Unified Extensible Firmware Interface |
| UML | Unified Modeling Language |
| UPS | Uninterruptible Power Supply |
| USB | Universal Serial Bus |
| V | Volt |
| VA | Volt Amps |
| VAR | Volt Amps Reactive |
| VDC | Volts Direct Current |
| VLSI | Very Large-Scale Integration |
| VRM | Voltage Regulator Module |

Table 96: List of abbreviations A2.8 – (W - $\psi$)

| Abbreviation Acronyms | Definition |
|---|---|
| **W** | **W**att |
| **WASP** | **W**ireless **A**pplication **S**ervice **P**rovider |
| **WEU** | **W**ater **E**conomizer **U**tilization |
| **Wh** | **W**att **h**our |
| **WSA** | **W**eighted **S**witching **A**ctivity |
| **XaaS** | Anything or Everything **as a S**ervice |
| $\boldsymbol{\beta}$ | **B**eta **I**ndex |
| $\boldsymbol{\theta}$ | Theta (ja/jm, jc, jb): Junction-to-Ambient, Junction-to-Moving Air, Junction-to-Case, Junction-to-Board |
| $\boldsymbol{\psi}$ | Psi (jt,jb): Junction-to-Top of package, Junction-to-Board |

# A3. Appendices

## A3a. Definition of Terms

***Power Triangle***

Power $P$ is an electrical level in Watt $[W]$ at a certain point in time. We distinguish power in true, effective, active, or real power. In common, power multiplies apparent power $S$ with reactive power $Q$, see (A3.1).

Apparent power $S$ is measured in units of volt amps $[VA]$ and is the product of voltage in volt $[V]$ and current in amps $[A]$, see (A3.2), calculated by using Ohm's Law. Power is directly proportional to voltage and current. The abbreviation $rms$ in this equation stands for root-mean-square (A3.3). A $rms$-based value is the amplitude of a signal divided by the square root.

Reactive power $Q$, also known as power factor $[PF]$ cosine $\varphi$[412], has the unit volt amps reactive $[VAR]$, see (A3.4) or Figure 158. The phase angle $\varphi$ is between voltage and current. The angle depends on different load types, such as capacitive, inductive, or resistive. Power is measured in watt.

**Equation 4: Definition – power, apparent power and power factor**

$$power\ [W] = apparent\ power * reactive\ power \qquad \text{(A3.1)}$$

$$apparent\ power[VA] = V_{rms}[V] * I_{rms}[A] \qquad \text{(A3.2)}$$

$$V_{rms} = \frac{V_{peak}}{\sqrt{2}} \qquad I_{rms} = \frac{I_{peak}}{\sqrt{2}} \qquad \text{(A3.3)}$$

$$reactive\ power\ [VAR] = \ power\ factor = cos(\varphi) \qquad \text{(A3.4)}$$

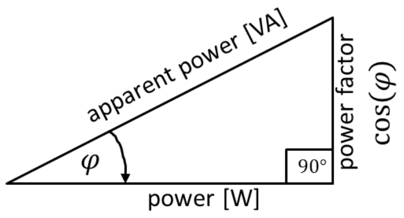


**Figure 158: Power triangle [Stö 2014]**

412 $\varphi$: phi

Under ideal circumstances, the phase angle $\varphi$ is zero, which means that no inductive, resistive, or inductive influences exist. Thus, the power factor cosine $\varphi$ is one. In this case, voltage and currents are “in phase”, cross the zero points at the same time, and the largest possible power occurs. Amplitudes of both are changing continuously either in Alternating or Direct Current (AC, DC) circuits. The AC power considers the different types of power, shown in the power triangle.

***Advanced Configuration and Power Interface Specification (ACPI)***

The advanced configuration and power interface (ACPI) specification defines the system states into:

- Global system states $(G)$
- Device power states $(D)$
- Sleeping states $(S)$
- Processor power states $(C)$
- Device and processor performance states $(P)$

Table 97 and Table 98 list the ACPI states and provides a rough description, which is interpreted by component vendors.

**Table 97: ACPI state definitions $(G, D, S)$ [HIM et al. 2013]**

| ACPI states | State | | Description |
|---|---|---|---|
| **Global system** | $\boldsymbol{G}$ | | **Global system state** |
| | $G3$ | | Mechanical off |
| | $G2$ | $S5$ | Soft off |
| | $G1$ | | Sleeping |
| | $G0$ | | Working |
| | | $S4$ | Non-volatile sleep |
| **Device power** | $\boldsymbol{D}$ | | **Device state** |
| | $D3$ | | Off |
| | $D3hot$ | | |
| | $D2$ | | Save more power |
| | $D1$ | | |
| | $D0$ | | Fully-on |
| **Sleeping** | $\boldsymbol{S}$ | | **Sleeping state** |
| | $S0$ | | |
| | $S1$ | | Low wake latency |
| | $S2$ | | |
| | $S3$ | | |
| | $S4$ | | Lowest power, longest wake latency |
| | $S5$ | | Soft off state |

Table 98: ACPI state definitions $(C, P)$ [HIM et al. 2013]

| ACPI states | State | Description |
|---|---|---|
| **Processor power** | $\boldsymbol{C}$ | **Processor power state** |
| | $C0$ | Executes instructions |
| | $C1$ | Lowest latency |
| | $C2$ | |
| | $C3$ | |
| **Device and processor performance** | $\boldsymbol{P}$ | **Performance state** |
| | $P0$ | Maximum performance |
| | $P1$ | |
| | $PN$ | Minimum performance |

## A3b. Overview of Various Metrics and Benchmarks in Their Related Domains

Table 99 and Table 100 summarize the various performance or power metrics as well as benchmarks that could be used in the data center, for a single rack enclosure, server system, or specialized on a certain chip.

Table 99: Metrics and benchmarks in various domains $(\boldsymbol{I})$

| Domain | Metrics | Benchmarks (tools) |
|---|---|---|
| **Data center** | Power usage effectiveness (PUE), data center infrastructure efficiency (DCiE), corporate average datacenter efficiency (CADE), airflow efficiency (AE), air economizer utilization (AEU), water economizer utilization (WEU), HVAC efficiency, cooling system sizing (CSS), data center cooling system efficiency (CSE), UPS losses, utilization (Load) factor, SWaP (Space, Watts and Performance), data center energy productivity (DCeP), green energy coefficient (GEC), energy reuse factor (ERF), carbon emission factor (CEF), carbon usage effectiveness (CUE), data center workload power efficiency (DWPE), supply heat index (SHI), return heat index (RHI), capture index (CI) | Calarch, Comis, DoE-2, EnergyPlus, Genopt |

Table 100: Metrics and benchmarks in various domains $(II)$

| Domain | Metrics | Benchmarks (tools) |
|---|---|---|
| **Rack enclosure** | PDU losses, IT or server equipment load density (W/sf), SWaP (space, watts and performance), return temperature index (RTI), rack cooling index (RCI), beta index $(\beta)$ | |
| **Server system, component** | IT equipment utilization (ITEU), IT equipment efficiency (ITEE), utilization (Load) factor, server utilization, green computing performance index (GCPI), peak performance (GFLOPS, TFLOPS), memory bandwidth (GB/sec), number of instructions / cycles, time period per job, 80 plus certificate (power supply), server utilization effectiveness (SUE) | Green 500, SPEC CPU, SPECpower, SPECviewperf, SPECwpc, SPECapc, SPEC ACCEL, SPEC MPI, SPEC OMP, SPEC HPC, SPECjAppServer, SPECjbb, SPECjvm, SPECmail, SPEC JVM, SPECvirt, SPECweb, LINPACK, STREAM, JouleSort, Server Efficiency Rating Tool (SERT),OCCT, Memtest, Iostat, IOzone, Iometer, Dbench, Hardinfo, GtkPerf, SysBench, Phoronix Test Suite, 3DMark, CPUBench, ProcessorMark, PassMark |
| **Chip** | Performance counter, number of instructions / cycles, theta / psi (thermal resistance) | Latbench, micro-Benchmarks |

## A3c. MATLAB Notation and Syntax – Classes, Labels, Usage, and Restrictions

The following section provides a short overview of MATLAB notations and syntax to provide the reader a better understanding of the terminology. MATLAB contains the classes: *double*, *character*, *string* or *cell* to format vectors, matrices, or arrays. Table 101 shows the relation between use cases and the corresponding classes.

Table 101: MATLAB relation between use cases and classes

| Use case | Class |
|---|---|
| Complete numbers | *Double* |
| Complete strings | *Character* or *string* |
| Mix of numbers and strings | *Cell* |

MATLAB labels internal variables by an equal sign. A label can contain numbers, underscores, and characters. A label cannot start with a number or contain a hyphen. A number includes a sign (+/-) followed by minimum of one numeric. Additionally, a dot divides decimal places. Equation 5 shows a brief overview of the characters, labels, numbers, and numbers within names.

Equation 5: Definition of MATLAB labels and numbers

$$\boldsymbol{numbersWithinNames} = \{\mathbf{0-9}, \boldsymbol{e}^{\pm\mathbf{0-9}}\},\ \boldsymbol{L}_{\boldsymbol{numbersWithinNames}} = \{\mathbf{0}, \mathbf{1}, \mathbf{9}, \mathbf{19}, \mathbf{199}, \mathbf{10}\boldsymbol{e}^{-\mathbf{3}} \dots\} \qquad \text{(A3.5)}$$

$$\boldsymbol{characters} = \{\_, \boldsymbol{a}-\boldsymbol{z}, \boldsymbol{A}-\boldsymbol{Z}\},\ \boldsymbol{L}_{\boldsymbol{characters}} = \{\_, \boldsymbol{a}, \_\boldsymbol{a}, \boldsymbol{a}\_, \boldsymbol{a}\_\boldsymbol{b}, \boldsymbol{A}, \boldsymbol{z}, \boldsymbol{Zz} \dots\} \qquad \text{(A3.6)}$$

$$\boldsymbol{label} = [\boldsymbol{characters}]^{+}[\boldsymbol{numbersWithinNames}]^{*}[\boldsymbol{characters}]^{*} \qquad \text{(A3.7)}$$

$$\boldsymbol{numbers} = [\pm]^{*}[\boldsymbol{numbersWithinNames}]^{+}[.]^{*}[\boldsymbol{numbersWithinNames}]^{*} \qquad \text{(A3.8)}$$

$$\boldsymbol{name\_number} = \{\boldsymbol{numbersWithinNames}|\boldsymbol{numbers}|\boldsymbol{labels}\} \qquad \text{(A3.9)}$$

The following terms describe the syntax of MATLAB vectors or matrices, whereby MATLAB defines a label in square brackets. A semicolon distinguishes between rows. Furthermore, a cell array requires additional curly brackets and single equation marks, shown in (A3.12). Percentage signs indicate user comments.

Equation 6: MATLAB row/column vector and matrix notation

$$[\boldsymbol{label}] = [\ [\boldsymbol{numbers}]^{+}\ [\boldsymbol{numbers}]^{*}\ ] \qquad \text{(A3.10)}$$

$$[\boldsymbol{label}] = [\ [\boldsymbol{numbers}]^{+};\ [\boldsymbol{numbers}]^{+}\ ] \qquad \text{(A3.11)}$$

$$[\boldsymbol{label}] = [\{\ [{}'[\boldsymbol{name\_number}]{}']^{+}\ \ [{}'[\boldsymbol{name\_number}]{}']^{*}\}\ ] \qquad \text{(A3.12)}$$

$$[label] = [\ \{\ [{}'[name\_number]{}']^{+}\ [{}'[name\_number]{}']^{*}\ ;\ [{}'[name\_number]{}']^{+}\ [{}'[name\_number]{}']^{*}\ \ \}\ ] \qquad \text{(A3.13)}$$

A numerical vector consists of a sequence of numbers within square brackets. Equation (A3.14) defines a single row vector (*Vector*), which includes the numeric values *one* and *two*.

```
Vector=[1 2]  equal to  Vector=[1, 2]                      (A3.14)

% 1 2
```

A row vector can be transformed to a column vector using a semicolon between both values.

```
Vector=[1; 2]                                               (A3.15)

% 1

% 2
```

Adding an additional row or column to our *Vector* ends up in a matrix $Ma$, shown in (A3.16). A matrix $Ma$ includes multiple vectors. The matrix dimension $(m\ x\ n)$ specifies the amount of rows $(m)$ and columns $(n)$. The complete numerical matrix $Ma$ can have different dimensions, for instance $(2x2, 2x3, 3x2)$.

```
Ma=[1 2; 3 4]  Ma=[1 2 3; 4 5 6]  Ma=[1 2; 3 4; 5 6]        (A3.16)

%   1 2                1 2 3              1 2

%   3 4                4 5 6              3 4

%                                         5 6
```

An index identifies a specific value from a vector or matrix. The first mandatory parameter determines a column for vectors and a row in the case of matrices. In matrices, the second parameter additionally identifies the column.

```
Ma=[1 2 3; 4 5 6]       Ma(1,3)            Ma(2,3)          (A3.17)

% 1 2 3                   3                  6

% 4 5 6
```

Generated vectors and matrices use the internal MATLAB class *double*, because they only contain numbers. The *cells* class is the equivalent for character or string arrays. Creating a *cell* array can be done in same manner, but the array requires curly brackets and single quotations.

```
Array = [{'one' 'two'}]                                     (A3.18)

% 'one' 'two'

Array = [{'one'; 'two'}]                                    (A3.19)

% 'one'

% 'two'
```

```
Array = [{'one' 'two'; 'three' 'four'}]                          (A3.20)
% 'one' 'two'
% 'three' 'four'
```

Equation (A3.21) mixes both data types, whereby the first row contains only numbers and the second row contains strings.

```
Mixed_array = [{1 2; 'three' 'four'}]                            (A3.21)
% [    1] [    2]
% 'three' 'four'
```

In the case of mixed classes, cell-specific MATLAB functions return failures. Therefore, it is necessary to convert strings to numbers or vice versa. One method is to store numbers within cells, which MATLAB shows with single quotation marks, and finally convert them.

```
Cell_array = [{'1' '2'; 'three' 'four'}]                         (A3.22)
%     '1'    '2'
% 'three' 'four'
```

Another variant is to save numbers separately from strings. We use an additional vector that includes only numbers. In our example (A3.23) low (*LV*), standard (*STD*), and future voltages are supported with levels from 1.35, 1.5, and 1.6 volt.

```
volt_possibilities=[{'voltage'}{'LV'}{'STD'}{'FUTURE'}]          (A3.23)
volt_values=[0 1.35 1.5 1.6]
```

We combine both in one-structure *volts* via a *for* loop, which includes all possible strings and values.

```
for all volts_possibilities                                      (A3.24)
      volts(i).possibilities  = volt_possibilities(i)
      volts(i).values         = volt_values(i)
end
```

If the value of string *LV* is required, we search the string within the *possibilities* of the *volts* structure. We save the index in case a string matches the search criteria. Afterwards, we provide the string value by using the same index at the *volts values* structure.

```
for all volts_possibilities                                    (A3.25)
    compare volts(i).possibilities with search string LV
    if match than remember index
    return
end
get value of string LV using volts(index).values
```

The *LV* string has an index *two* and thus we address the content using the index and the vector name, which contains the values.

```
volts(2)                                                       (A3.26)
% possibilities: {'LV'}
% values: 1.3500
```

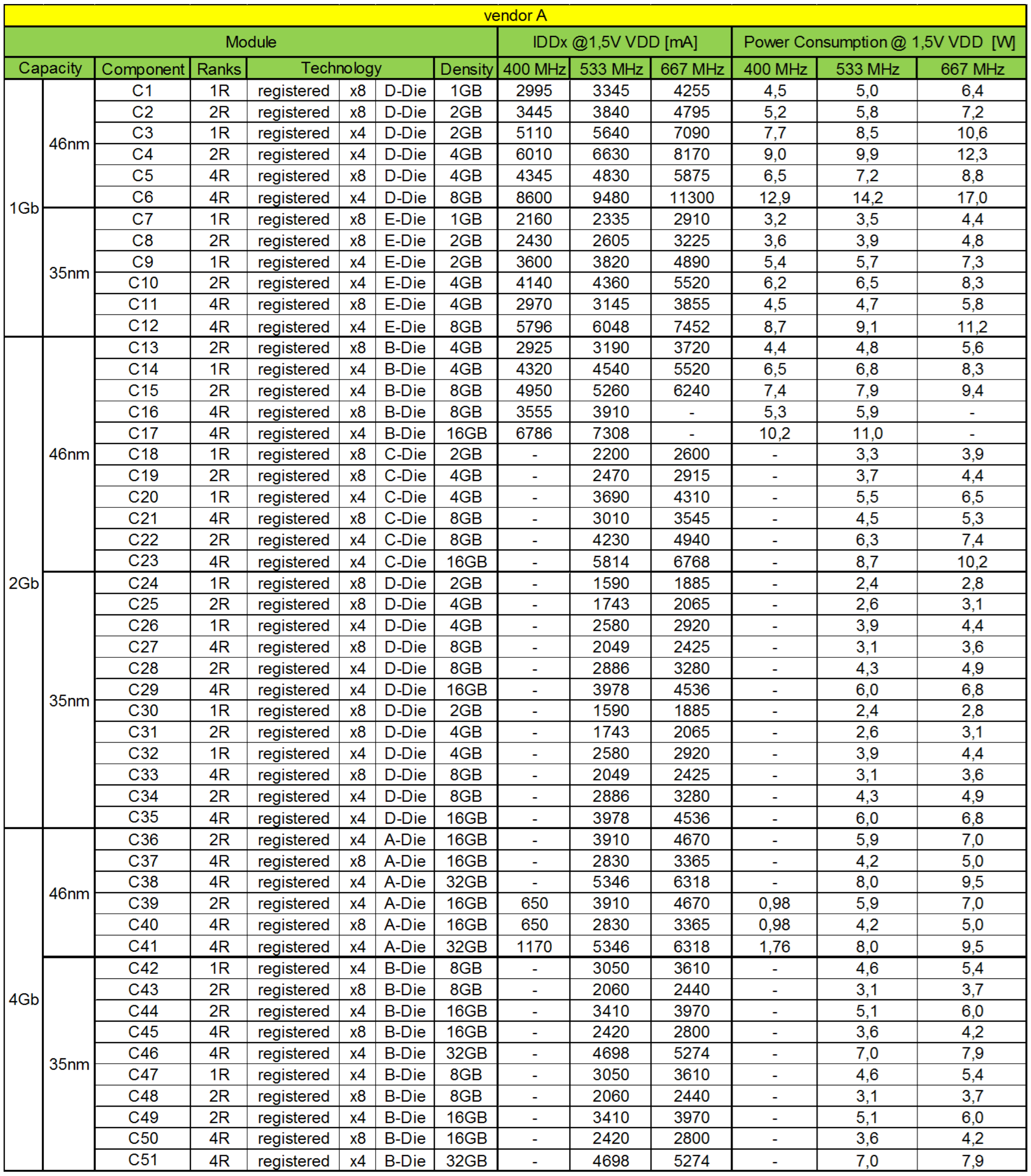
```
volts(2).values                                                (A3.27)
% 1.3500
```

Data adaption or extension is easily manageable by changing or adding new values in related vectors. MATLAB provides a huge range of class-specific functions, such as converting from string to number *str2num* and other common functionalities. The MathWorks homepage[413] provides further information.

## A3d. Memory Module Analyzation and Characteristics $(C1 - C78)$

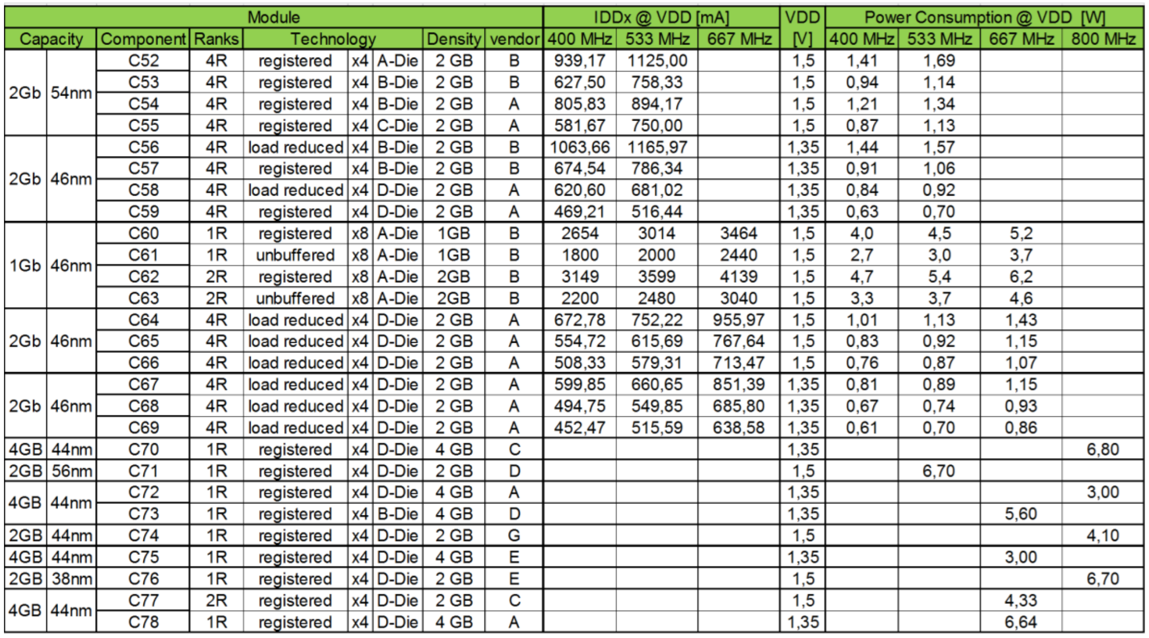
Figure 159 presents the memory module characteristics, currents $[mA]$, and power consumptions $[W]$ of our measurements, which we respectively analyze for the memory modules of vendor $A$. Additionally, we observe the measurement results considering the vendors $[A - G]$, as shown in Figure 160.

[413] MathWorks homepage: http://www.mathworks.com/help/index.html



| vendor A | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Module | | | | | | | | IDDx @1,5V VDD [mA] | | | Power Consumption @ 1,5V VDD [W] | | |
| Capacity | | Component | Ranks | Technology | | | Density | 400 MHz | 533 MHz | 667 MHz | 400 MHz | 533 MHz | 667 MHz |
| 1Gb | 46nm | C1 | 1R | registered | x8 | D-Die | 1GB | 2995 | 3345 | 4255 | 4,5 | 5,0 | 6,4 |
| | | C2 | 2R | registered | x8 | D-Die | 2GB | 3445 | 3840 | 4795 | 5,2 | 5,8 | 7,2 |
| | | C3 | 1R | registered | x4 | D-Die | 2GB | 5110 | 5640 | 7090 | 7,7 | 8,5 | 10,6 |
| | | C4 | 2R | registered | x4 | D-Die | 4GB | 6010 | 6630 | 8170 | 9,0 | 9,9 | 12,3 |
| | | C5 | 4R | registered | x8 | D-Die | 4GB | 4345 | 4830 | 5875 | 6,5 | 7,2 | 8,8 |
| | | C6 | 4R | registered | x4 | D-Die | 8GB | 8600 | 9480 | 11300 | 12,9 | 14,2 | 17,0 |
| | 35nm | C7 | 1R | registered | x8 | E-Die | 1GB | 2160 | 2335 | 2910 | 3,2 | 3,5 | 4,4 |
| | | C8 | 2R | registered | x8 | E-Die | 2GB | 2430 | 2605 | 3225 | 3,6 | 3,9 | 4,8 |
| | | C9 | 1R | registered | x4 | E-Die | 2GB | 3600 | 3820 | 4890 | 5,4 | 5,7 | 7,3 |
| | | C10 | 2R | registered | x4 | E-Die | 4GB | 4140 | 4360 | 5520 | 6,2 | 6,5 | 8,3 |
| | | C11 | 4R | registered | x8 | E-Die | 4GB | 2970 | 3145 | 3855 | 4,5 | 4,7 | 5,8 |
| | | C12 | 4R | registered | x4 | E-Die | 8GB | 5796 | 6048 | 7452 | 8,7 | 9,1 | 11,2 |
| 2Gb | 46nm | C13 | 2R | registered | x8 | B-Die | 4GB | 2925 | 3190 | 3720 | 4,4 | 4,8 | 5,6 |
| | | C14 | 1R | registered | x4 | B-Die | 4GB | 4320 | 4540 | 5520 | 6,5 | 6,8 | 8,3 |
| | | C15 | 2R | registered | x4 | B-Die | 8GB | 4950 | 5260 | 6240 | 7,4 | 7,9 | 9,4 |
| | | C16 | 4R | registered | x8 | B-Die | 8GB | 3555 | 3910 | - | 5,3 | 5,9 | - |
| | | C17 | 4R | registered | x4 | B-Die | 16GB | 6786 | 7308 | - | 10,2 | 11,0 | - |
| | | C18 | 1R | registered | x8 | C-Die | 2GB | - | 2200 | 2600 | - | 3,3 | 3,9 |
| | | C19 | 2R | registered | x8 | C-Die | 4GB | - | 2470 | 2915 | - | 3,7 | 4,4 |
| | | C20 | 1R | registered | x4 | C-Die | 4GB | - | 3690 | 4310 | - | 5,5 | 6,5 |
| | | C21 | 4R | registered | x8 | C-Die | 8GB | - | 3010 | 3545 | - | 4,5 | 5,3 |
| | | C22 | 2R | registered | x4 | C-Die | 8GB | - | 4230 | 4940 | - | 6,3 | 7,4 |
| | | C23 | 4R | registered | x4 | C-Die | 16GB | - | 5814 | 6768 | - | 8,7 | 10,2 |
| | 35nm | C24 | 1R | registered | x8 | D-Die | 2GB | - | 1590 | 1885 | - | 2,4 | 2,8 |
| | | C25 | 2R | registered | x8 | D-Die | 4GB | - | 1743 | 2065 | - | 2,6 | 3,1 |
| | | C26 | 1R | registered | x4 | D-Die | 4GB | - | 2580 | 2920 | - | 3,9 | 4,4 |
| | | C27 | 4R | registered | x8 | D-Die | 8GB | - | 2049 | 2425 | - | 3,1 | 3,6 |
| | | C28 | 2R | registered | x4 | D-Die | 8GB | - | 2886 | 3280 | - | 4,3 | 4,9 |
| | | C29 | 4R | registered | x4 | D-Die | 16GB | - | 3978 | 4536 | - | 6,0 | 6,8 |
| | | C30 | 1R | registered | x8 | D-Die | 2GB | - | 1590 | 1885 | - | 2,4 | 2,8 |
| | | C31 | 2R | registered | x8 | D-Die | 4GB | - | 1743 | 2065 | - | 2,6 | 3,1 |
| | | C32 | 1R | registered | x4 | D-Die | 4GB | - | 2580 | 2920 | - | 3,9 | 4,4 |
| | | C33 | 4R | registered | x8 | D-Die | 8GB | - | 2049 | 2425 | - | 3,1 | 3,6 |
| | | C34 | 2R | registered | x4 | D-Die | 8GB | - | 2886 | 3280 | - | 4,3 | 4,9 |
| | | C35 | 4R | registered | x4 | D-Die | 16GB | - | 3978 | 4536 | - | 6,0 | 6,8 |
| 4Gb | 46nm | C36 | 2R | registered | x4 | A-Die | 16GB | - | 3910 | 4670 | - | 5,9 | 7,0 |
| | | C37 | 4R | registered | x8 | A-Die | 16GB | - | 2830 | 3365 | - | 4,2 | 5,0 |
| | | C38 | 4R | registered | x4 | A-Die | 32GB | - | 5346 | 6318 | - | 8,0 | 9,5 |
| | | C39 | 2R | registered | x4 | A-Die | 16GB | 650 | 3910 | 4670 | 0,98 | 5,9 | 7,0 |
| | | C40 | 4R | registered | x8 | A-Die | 16GB | 650 | 2830 | 3365 | 0,98 | 4,2 | 5,0 |
| | | C41 | 4R | registered | x4 | A-Die | 32GB | 1170 | 5346 | 6318 | 1,76 | 8,0 | 9,5 |
| | 35nm | C42 | 1R | registered | x4 | B-Die | 8GB | - | 3050 | 3610 | - | 4,6 | 5,4 |
| | | C43 | 2R | registered | x8 | B-Die | 8GB | - | 2060 | 2440 | - | 3,1 | 3,7 |
| | | C44 | 2R | registered | x4 | B-Die | 16GB | - | 3410 | 3970 | - | 5,1 | 6,0 |
| | | C45 | 4R | registered | x8 | B-Die | 16GB | - | 2420 | 2800 | - | 3,6 | 4,2 |
| | | C46 | 4R | registered | x4 | B-Die | 32GB | - | 4698 | 5274 | - | 7,0 | 7,9 |
| | | C47 | 1R | registered | x4 | B-Die | 8GB | - | 3050 | 3610 | - | 4,6 | 5,4 |
| | | C48 | 2R | registered | x8 | B-Die | 8GB | - | 2060 | 2440 | - | 3,1 | 3,7 |
| | | C49 | 2R | registered | x4 | B-Die | 16GB | - | 3410 | 3970 | - | 5,1 | 6,0 |
| | | C50 | 4R | registered | x8 | B-Die | 16GB | - | 2420 | 2800 | - | 3,6 | 4,2 |
| | | C51 | 4R | registered | x4 | B-Die | 32GB | - | 4698 | 5274 | - | 7,0 | 7,9 |

**Figure 159: Memory module characteristics of vendor $A$ $(C1, C51)$ – measurements**



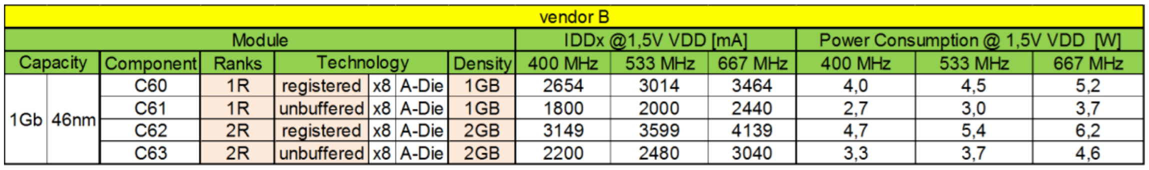
| Module | | | | | | | | | IDDx @ VDD [mA] | | | VDD | Power Consumption @ VDD [W] | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Capacity | | Component | Ranks | Technology | | | Density | vendor | 400 MHz | 533 MHz | 667 MHz | [V] | 400 MHz | 533 MHz | 667 MHz | 800 MHz |
| 2Gb | 54nm | C52 | 4R | registered | x4 | A-Die | 2 GB | B | 939,17 | 1125,00 | | 1,5 | 1,41 | 1,69 | | |
| | | C53 | 4R | registered | x4 | B-Die | 2 GB | B | 627,50 | 758,33 | | 1,5 | 0,94 | 1,14 | | |
| | | C54 | 4R | registered | x4 | B-Die | 2 GB | A | 805,83 | 894,17 | | 1,5 | 1,21 | 1,34 | | |
| | | C55 | 4R | registered | x4 | C-Die | 2 GB | A | 581,67 | 750,00 | | 1,5 | 0,87 | 1,13 | | |
| 2Gb | 46nm | C56 | 4R | load reduced | x4 | B-Die | 2 GB | B | 1063,66 | 1165,97 | | 1,35 | 1,44 | 1,57 | | |
| | | C57 | 4R | registered | x4 | B-Die | 2 GB | B | 674,54 | 786,34 | | 1,35 | 0,91 | 1,06 | | |
| | | C58 | 4R | load reduced | x4 | D-Die | 2 GB | A | 620,60 | 681,02 | | 1,35 | 0,84 | 0,92 | | |
| | | C59 | 4R | registered | x4 | D-Die | 2 GB | A | 469,21 | 516,44 | | 1,35 | 0,63 | 0,70 | | |
| 1Gb | 46nm | C60 | 1R | registered | x8 | A-Die | 1GB | B | 2654 | 3014 | 3464 | 1,5 | 4,0 | 4,5 | 5,2 | |
| | | C61 | 1R | unbuffered | x8 | A-Die | 1GB | B | 1800 | 2000 | 2440 | 1,5 | 2,7 | 3,0 | 3,7 | |
| | | C62 | 2R | registered | x8 | A-Die | 2GB | B | 3149 | 3599 | 4139 | 1,5 | 4,7 | 5,4 | 6,2 | |
| | | C63 | 2R | unbuffered | x8 | A-Die | 2GB | B | 2200 | 2480 | 3040 | 1,5 | 3,3 | 3,7 | 4,6 | |
| 2Gb | 46nm | C64 | 4R | load reduced | x4 | D-Die | 2 GB | A | 672,78 | 752,22 | 955,97 | 1,5 | 1,01 | 1,13 | 1,43 | |
| | | C65 | 4R | load reduced | x4 | D-Die | 2 GB | A | 554,72 | 615,69 | 767,64 | 1,5 | 0,83 | 0,92 | 1,15 | |
| | | C66 | 4R | load reduced | x4 | D-Die | 2 GB | A | 508,33 | 579,31 | 713,47 | 1,5 | 0,76 | 0,87 | 1,07 | |
| 2Gb | 46nm | C67 | 4R | load reduced | x4 | D-Die | 2 GB | A | 599,85 | 660,65 | 851,39 | 1,35 | 0,81 | 0,89 | 1,15 | |
| | | C68 | 4R | load reduced | x4 | D-Die | 2 GB | A | 494,75 | 549,85 | 685,80 | 1,35 | 0,67 | 0,74 | 0,93 | |
| | | C69 | 4R | load reduced | x4 | D-Die | 2 GB | A | 452,47 | 515,59 | 638,58 | 1,35 | 0,61 | 0,70 | 0,86 | |
| 4GB | 44nm | C70 | 1R | registered | x4 | D-Die | 4 GB | C | | | | 1,35 | | | | 6,80 |
| 2GB | 56nm | C71 | 1R | registered | x4 | D-Die | 2 GB | D | | | | 1,5 | | 6,70 | | |
| 4GB | 44nm | C72 | 1R | registered | x4 | D-Die | 4 GB | A | | | | 1,35 | | | | 3,00 |
| | | C73 | 1R | registered | x4 | B-Die | 4 GB | D | | | | 1,35 | | | 5,60 | |
| 2GB | 44nm | C74 | 1R | registered | x4 | D-Die | 2 GB | G | | | | 1,5 | | | | 4,10 |
| 4GB | 44nm | C75 | 1R | registered | x4 | D-Die | 4 GB | E | | | | 1,35 | | | 3,00 | |
| 2GB | 38nm | C76 | 1R | registered | x4 | D-Die | 2 GB | E | | | | 1,5 | | | | 6,70 |
| 4GB | 44nm | C77 | 2R | registered | x4 | D-Die | 2 GB | C | | | | 1,5 | | | 4,33 | |
| | | C78 | 1R | registered | x4 | D-Die | 4 GB | A | | | | 1,35 | | | 6,64 | |

**Figure 160: Memory module characteristics of diverse vendors $(C52, C78)$ – measurements**

We exemplarily describe our results of the memory modules of the vendor $A$ considering the capacity, fabrication size, ranks, technology, and density, as shown in Figure 159. We measure the memory current $IDD\ [mA]$[414] and calculate the power consumption in $[W]$ of the components $C1 - C51$ with a constant supply voltage $VDD = 1.5V$ concerning the frequencies $f = \{800MHz, 1066MHz, 1333MHz\}$ using an external hardware adapter. We present our analysis results considering the synchronization mode, vendor, and frequency, and finally show the dependencies of the capacity, ranks, technology, and density.

### *Synchronization Mode*

An unbuffered memory module consumes always less power compared to a registered module with the same technical specification. Our internal results show that the power consumption of two modules with equivalent technical specification[415] differs up to ten percent.



| vendor B | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Module | | | | | | | | IDDx @1,5V VDD [mA] | | | Power Consumption @ 1,5V VDD [W] | | |
| Capacity | | Component | Ranks | Technology | | | Density | 400 MHz | 533 MHz | 667 MHz | 400 MHz | 533 MHz | 667 MHz |
| 1Gb | 46nm | C60 | 1R | registered | x8 | A-Die | 1GB | 2654 | 3014 | 3464 | 4,0 | 4,5 | 5,2 |
| | | C61 | 1R | unbuffered | x8 | A-Die | 1GB | 1800 | 2000 | 2440 | 2,7 | 3,0 | 3,7 |
| | | C62 | 2R | registered | x8 | A-Die | 2GB | 3149 | 3599 | 4139 | 4,7 | 5,4 | 6,2 |
| | | C63 | 2R | unbuffered | x8 | A-Die | 2GB | 2200 | 2480 | 3040 | 3,3 | 3,7 | 4,6 |

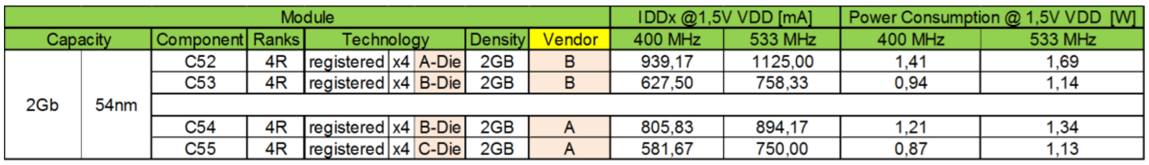
**Figure 161: Memory modules – synchronization mode**

[414] Current $IDD$: drain-current of a CMOS circuit

[415] Memory modules: vendor $B$ DDR3-SDRAM 1GB 1R A (registered vs. unbuffered)

***Vendor***

In general, we study all characteristic combinations of each category separately to identify their relevance and find the weight coefficients. We analyze single characteristics, such as the memory vendor, which can be another cause for variations in the power consumption of the same specified module.



| Module | | | | | | | | | IDDx @1,5V VDD [mA] | | Power Consumption @ 1,5V VDD [W] | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Capacity | | Component | Ranks | Technology | | | Density | Vendor | 400 MHz | 533 MHz | 400 MHz | 533 MHz |
| 2Gb | 54nm | C52 | 4R | registered | x4 | A-Die | 2GB | B | 939,17 | 1125,00 | 1,41 | 1,69 |
| | | C53 | 4R | registered | x4 | B-Die | 2GB | B | 627,50 | 758,33 | 0,94 | 1,14 |
| | | | | | | | | | | | | |
| | | C54 | 4R | registered | x4 | B-Die | 2GB | A | 805,83 | 894,17 | 1,21 | 1,34 |
| | | C55 | 4R | registered | x4 | C-Die | 2GB | A | 581,67 | 750,00 | 0,87 | 1,13 |

Figure 162: Memory modules – vendor

***Frequency***

Furthermore, we found that a frequency change from $f_1 = 800MHz$ to $f_2 = 1066MHz$ increases the power in the mean nearly eight percent. A changing frequency, from $f_2 = 1066MHz$ to $f_3 = 1333MHz$, results in 19 percent higher power consumption.

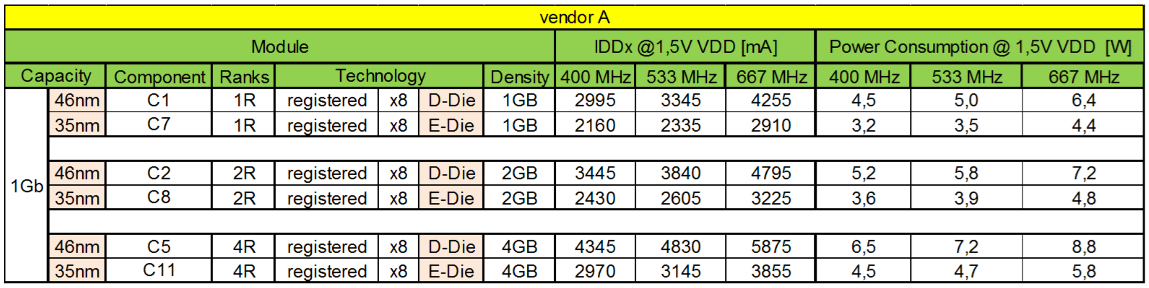
Table 102: Memory modules – frequency

| Frequency change | Increase [%] |
|---|---|
| $f_1 = 800MHz$ -> $f_2 = 1066MHz$ | 8 |
| $f_2 = 1066MHz$ -> $f_3 = 1333MHz$ | 19 |
| $f_3 = 1333MHz$ -> $f_4 = 1600MHz$ | 4 |
| $f_4 = 1600MHz$ -> $f_5 = 1866MHz$ | 3 |

***Capacity, Ranks, Technology, and Density***

Figure 163 addresses the differences in current $[mA]$ between the fabrication size and die technology[416], whereby all other characteristics are fixed. We set the fabrication size from $46nm$ to $35nm$, and set the die from $D$ to $E$. The power consumption is approximately 32 percent lower for a $(C7, C8, C11 - 35nm, E)$ module than for a $(C1, C2, C5 - 46nm, D)$ module, see pairs of $(C1, C7; C2, C8; C5, C11)$. The decrease of a $(C5, C11 - 4GB, 4R)$ module is four percent larger than for a $(C1, C7 - 1GB, 1R)$ module, which we neglect in our approach.

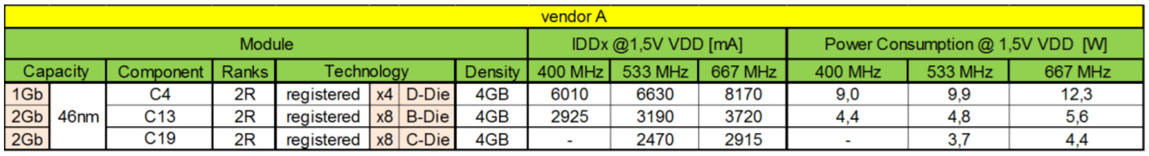
[416] Die technology: component revision, denote by a letter $\{A, B, C, D, E, F, G, H, J, L, M, N\}$



| vendor A | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Module | | | | | | | | IDDx @1,5V VDD [mA] | | | Power Consumption @ 1,5V VDD [W] | | |
| Capacity | | Component | Ranks | Technology | | | Density | 400 MHz | 533 MHz | 667 MHz | 400 MHz | 533 MHz | 667 MHz |
| 1Gb | 46nm | C1 | 1R | registered | x8 | D-Die | 1GB | 2995 | 3345 | 4255 | 4,5 | 5,0 | 6,4 |
| | 35nm | C7 | 1R | registered | x8 | E-Die | 1GB | 2160 | 2335 | 2910 | 3,2 | 3,5 | 4,4 |
| | | | | | | | | | | | | | |
| | 46nm | C2 | 2R | registered | x8 | D-Die | 2GB | 3445 | 3840 | 4795 | 5,2 | 5,8 | 7,2 |
| | 35nm | C8 | 2R | registered | x8 | E-Die | 2GB | 2430 | 2605 | 3225 | 3,6 | 3,9 | 4,8 |
| | | | | | | | | | | | | | |
| | 46nm | C5 | 4R | registered | x8 | D-Die | 4GB | 4345 | 4830 | 5875 | 6,5 | 7,2 | 8,8 |
| | 35nm | C11 | 4R | registered | x8 | E-Die | 4GB | 2970 | 3145 | 3855 | 4,5 | 4,7 | 5,8 |

Figure 163: Memory modules – fabrication size and die

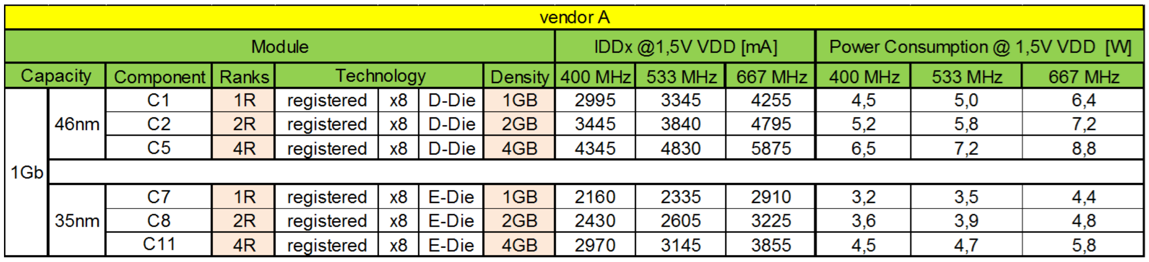
We analyze the memory capacity, rank linking[417] (e.g. x4, x8, x16) technology, and die type. If we have twice the capacity and the rank linking, the die type is the significant factor. We compare the components $(C4, C13)$ with a reduced power by 52 percent from $D$ to $B$ die. We see a decrease of approximately 62 percent when we compare the $D$ and $C$ die. The effect the frequency has on the power consumption is negligible.



| vendor A | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Module | | | | | | | | IDDx @1,5V VDD [mA] | | | Power Consumption @ 1,5V VDD [W] | | |
| Capacity | | Component | Ranks | Technology | | | Density | 400 MHz | 533 MHz | 667 MHz | 400 MHz | 533 MHz | 667 MHz |
| 1Gb | 46nm | C4 | 2R | registered | x4 | D-Die | 4GB | 6010 | 6630 | 8170 | 9,0 | 9,9 | 12,3 |
| 2Gb | | C13 | 2R | registered | x8 | B-Die | 4GB | 2925 | 3190 | 3720 | 4,4 | 4,8 | 5,6 |
| 2Gb | | C19 | 2R | registered | x8 | C-Die | 4GB | - | 2470 | 2915 | - | 3,7 | 4,4 |

Figure 164: Memory modules – capacity, rank linking, and die

Figure 165 shows the memory modules with twofold density and ranks. A doubled density from $(C1, C7 - 1GB, 1R)$ to $(C2, C8 - 2GB, 2R)$, as compared with the pairs $(C1, C2; C7, C8)$, results in a 13 percent power increase. We see a 23 percent rise for a $(C2, C8 - 2GB, 2R)$ to a $(C5, C11 - 4GB, 4R)$ module, such as $(C2, C5; C8, C11)$. The effect the frequency has on the power consumption is negligible.

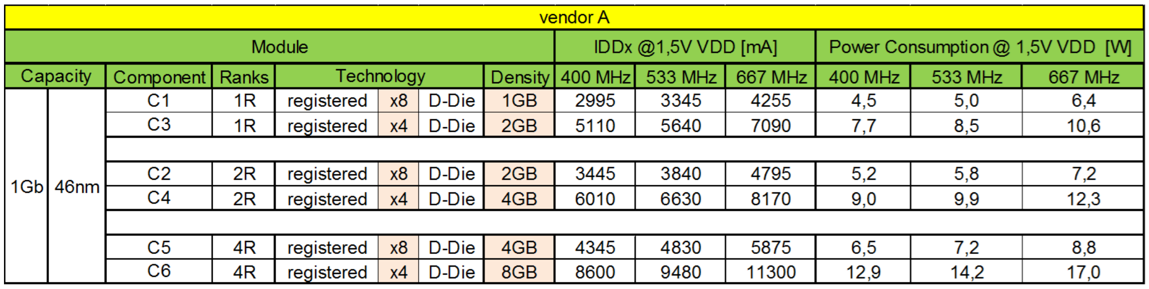


| vendor A | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Module | | | | | | | | IDDx @1,5V VDD [mA] | | | Power Consumption @ 1,5V VDD [W] | | |
| Capacity | | Component | Ranks | Technology | | | Density | 400 MHz | 533 MHz | 667 MHz | 400 MHz | 533 MHz | 667 MHz |
| 1Gb | 46nm | C1 | 1R | registered | x8 | D-Die | 1GB | 2995 | 3345 | 4255 | 4,5 | 5,0 | 6,4 |
| | | C2 | 2R | registered | x8 | D-Die | 2GB | 3445 | 3840 | 4795 | 5,2 | 5,8 | 7,2 |
| | | C5 | 4R | registered | x8 | D-Die | 4GB | 4345 | 4830 | 5875 | 6,5 | 7,2 | 8,8 |
| | | | | | | | | | | | | | |
| | 35nm | C7 | 1R | registered | x8 | E-Die | 1GB | 2160 | 2335 | 2910 | 3,2 | 3,5 | 4,4 |
| | | C8 | 2R | registered | x8 | E-Die | 2GB | 2430 | 2605 | 3225 | 3,6 | 3,9 | 4,8 |
| | | C11 | 4R | registered | x8 | E-Die | 4GB | 2970 | 3145 | 3855 | 4,5 | 4,7 | 5,8 |

Figure 165: Memory modules – ranks and density

Figure 166 summarizes the modules with double density, but halved rank linking. The power increases approximately 69 percent from a $(C1 - 1GB, x8)$ to a $(C3 - 2GB, x4)$ module, which is comparable by a rise of 72 percent from $(C2 - 2GB, x8)$ up to $(C4 - 4GB, x4)$. The power consumption of the memory pair $(C5 - 4GB, x8)$ and $(C6 - 8GB, x4)$ nearly doubles from $C5$ to $C6$. Furthermore, we see a frequency influence of nearly two percent from a lower to an upper frequency. We conclude that the memory factors ranks, rank linking, die, and density rely on each other. Therefore, we handle them as a linear system of equations.

[417] Rank linking: number of chip's output pins / bit wide



| vendor A | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Module | | | | | | | | IDDx @1,5V VDD [mA] | | | Power Consumption @ 1,5V VDD [W] | | |
| Capacity | | Component | Ranks | Technology | | | Density | 400 MHz | 533 MHz | 667 MHz | 400 MHz | 533 MHz | 667 MHz |
| 1Gb | 46nm | C1 | 1R | registered | x8 | D-Die | 1GB | 2995 | 3345 | 4255 | 4,5 | 5,0 | 6,4 |
| | | C3 | 1R | registered | x4 | D-Die | 2GB | 5110 | 5640 | 7090 | 7,7 | 8,5 | 10,6 |
| | | | | | | | | | | | | | |
| | | C2 | 2R | registered | x8 | D-Die | 2GB | 3445 | 3840 | 4795 | 5,2 | 5,8 | 7,2 |
| | | C4 | 2R | registered | x4 | D-Die | 4GB | 6010 | 6630 | 8170 | 9,0 | 9,9 | 12,3 |
| | | | | | | | | | | | | | |
| | | C5 | 4R | registered | x8 | D-Die | 4GB | 4345 | 4830 | 5875 | 6,5 | 7,2 | 8,8 |
| | | C6 | 4R | registered | x4 | D-Die | 8GB | 8600 | 9480 | 11300 | 12,9 | 14,2 | 17,0 |

Figure 166: Memory modules – ranks (x) and density

## A3e. Detailed Memory Module Characteristics $(C14, C24, C26, C70 - C78)$

In this section, we present the memory module characteristics that we use in our simulation and evaluation.

Table 103: Detailed memory characteristics $(C70, C71)$

| Memory characteristics | Memory module $(C70)$ | Memory module $(C71)$ |
|---|---|---|
| Vendor | `{'Micron'}; {'C'}` | `{'Qimonda'}; {'D'}` |
| Capacity (size) [GB] | `{'4GB'}` | `{'2GB'}` |
| Density [GB] | `{'4GB'}` | `{'2GB'}` |
| Die (component revision) | `{'D'}` | `{'D'}` |
| Fabrication size [nm] | `{'44nm'}` | `{'56nm'}` |
| Synchronization mode | `{'registered'}` | `{'registered'}` |
| Module ranks, rank linking (data width) | `{'1R'}; {'SR'}, {'x4'}` | `{'1R'}; {'SR'}, {'x4'}` |
| Timings | `{'11'}` | `{'7'}` |
| Error correction | `{'ECC'}` | `{'ECC'}` |
| Frequency [MHz] | `{'800'}` | `{'533'}` |
| Voltage [VDC] [418] | `{'LV'};` 1.35VDC | `{'STD'};` 1.5VDC |
| Transfer rate / throughput [MHz] | `{'1600'}; {'PC3-12800'}` | `{'1066'}, {'PC3-8500R'}` |

[418] VDC: volts direct current

Table 104: Detailed memory characteristics $(C26, C74, C76)$

| Memory characteristics | Memory module $(C26)$ | Memory module $(C74)$ | Memory module $(C76)$ |
|---|---|---|---|
| Vendor | {'A'} | {'G'} | {'E'} |
| Capacity (size) [GB] | {'2GB'} | {'2GB'} | {'2GB'} |
| Density [GB] | {'4GB'} | {'2GB'} | {'2GB'} |
| Die (component revision) | {'D'} | {'D'} | {'D'} |
| Fabrication size [nm] | {'35nm'} | {'44nm'} | {'38nm'} |
| Synchronization mode | {'registered'} | {'registered'} | {'registered'} |
| Module ranks, rank linking (data width) | {'1R'}; {'SR'}, {'x4'} | {'1R'}; {'SR'}, {'x4'} | {'1R'}; {'SR'}, {'x4'} |
| Timings | {'11'} | {'11'} | {'11'} |
| Error correction | {''} | {'ECC'} | {'ECC'} |
| Frequency [MHz] | {'533'} | {'800'} | {'800'} |
| Voltage [VDC] [419] | {'STD'}; 1.5VDC | {'STD'}; 1.5VDC | {'STD'}; 1.5VDC |
| Transfer rate / throughput [MHz] | {'1066'}; {'PC3-8500'} | {'1600'}; {'PC3-12800'} | {'1600'}; {'PC3-12800'} |

[419] VDC: volts direct current

Table 105: Detailed memory characteristics $(C72, C73, C75)$

| Memory characteristics | Memory module $(C72)$ | Memory module $(C73)$ | Memory module $(C75)$ |
|---|---|---|---|
| Vendor | {'A'} | {'Qimonda'}; {'D'} | {'E'} |
| Capacity (size) [GB] | {'4GB'} | {'4GB'} | {'4GB'} |
| Density [GB] | {'4GB'} | {'4GB'} | {'4GB'} |
| Die (component revision) | {'D'} | {'B'} | {'D'} |
| Fabrication size [nm] | {'44nm'} | {'44nm'} | {'44nm'} |
| Synchronization mode | {'registered'} | {'registered'} | {'registered'} |
| Module ranks, rank linking (data width) | {'1R'}; {'SR'}, {'x4'} | {'1R'}; {'SR'}, {'x4'} | {'1R'}; {'SR'}, {'x4'} |
| Timings | {'11'} | {'11'} | {'11'} |
| Error correction | {'ECC'} | {'ECC'} | {'ECC'} |
| Frequency [MHz] | {'800'} | {'667'} | {'667'} |
| Voltage [VDC] [420] | {'LV'}; 1.35VDC | {'LV'}; 1.35VDC | {'LV'}; 1.35VDC |
| Transfer rate / throughput [MHz] | {'1600'}; {'PC3-12800'} | {'1333'}, {'PC3-10600'} | {'1333'}, {'PC3-10600'} |

Table 106: Detailed memory characteristics $(C24, C26)$

| Memory characteristics | Memory module $(C24)$ | Memory module $(C26)$ |
|---|---|---|
| Vendor | {'A'} | {'A'} |
| Capacity (size) [GB] | {'2GB'} | {'2GB'} |
| Density [GB] | {'2GB'} | {'4GB'} |
| Die (component revision) | {'D'} | {'D'} |
| Fabrication size [nm] | {'35nm'} | {'35nm'} |
| Synchronization mode | {'registered'} | {'registered'} |
| Module ranks, rank linking (data width) | {'2R'}; {'DR'}, {'x4'} | {'1R'}; {'SR'}, {'x4'} |
| Timings | {'11'} | {'11'} |
| Error correction | {'ECC'} | {''} |
| Frequency [MHz] | {'800'} | {'533'} |
| Voltage [VDC] | {'STD'}; 1.5VDC | {'STD'}; 1.5VDC |
| Transfer rate / throughput [MHz] | {'1600'}; {'PC3-12800'} | {'1066'}; {'PC3-8500'} |

[420] VDC: volts direct current

Table 107: Detailed memory characteristics $(C14, C20, C26)$

| Memory characteristics | Memory module $(C14)$ | Memory module $(C20)$ | Memory module $(C26)$ |
|---|---|---|---|
| Vendor | {'A'} | {'A'} | {'A'} |
| Capacity (size) [GB] | {'2GB'} | {'2GB'} | {'2GB'} |
| Density [GB] | {'4GB'} | {'4GB'} | {'4GB'} |
| Die (component revision) | {'B'} | {'C'} | {'D'} |
| Fabrication size [nm] | {'46nm'} | {'46nm'} | {'35nm'} |
| Synchronization mode | {'registered'} | {'registered'} | {'registered'} |
| Module ranks, rank linking (data width) | {'1R'}; {'SR'}, {'x4'} | {'1R'}; {'SR'}, {'x4'} | {'1R'}; {'SR'}, {'x4'} |
| Timings | {'11'} | {'11'} | {'11'} |
| Error correction | {''} | {''} | {''} |
| Frequency [MHz] | {'533'} | {'533'} | {'533'} |
| Voltage [VDC] [421] | {'STD'}; 1.5VDC | {'STD'}; 1.5VDC | {'STD'}; 1.5VDC |
| Transfer rate / throughput [MHz] | {'1066'}, {'PC3-8500'} | {'1066'}, {'PC3-8500R'} | {'1066'}; {'PC3-8500'} |

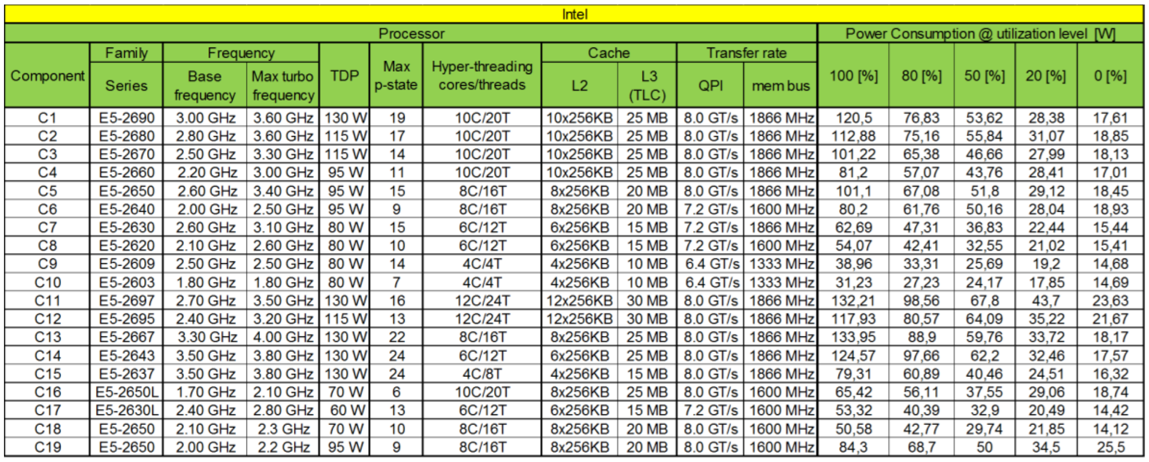
Table 108: Detailed memory characteristics $(C77, C78)$

| Memory characteristics | Memory module $(C77)$ | Memory module $(C78)$ |
|---|---|---|
| Vendor | {'Micron'}; {'C'} | {'A'} |
| Capacity (size) [GB] | {'4GB'} | {'4GB'} |
| Density [GB] | {'2GB'} | {'4GB'} |
| Die (component revision) | {'D'} | {'D'} |
| Fabrication size [nm] | {'44nm'} | {'44nm'} |
| Synchronization mode | {'registered'} | {'registered'} |
| Module ranks, rank linking (data width) | {'2R'}; {'DR'}, {'x4'} | {'1R'}; {'SR'}, {'x4'} |
| Timings | {'9'} | {'7'} |
| Error correction | {'ECC'} | {''} |
| Frequency [MHz] | {'800'} | {'667'} |
| Voltage [VDC] | {'STD'}; 1.5VDC | {'LV'}; 1.35VDC |
| Transfer rate / throughput [MHz] | {'1600'}; {'PC3-12800'} | {'1333'}, {'PC3-10600'} |

[421] VDC: volts direct current

## A3f. Processor Analyzation and Characteristics $(C1 - C19)$

Figure 167 lists the processor characteristics and power consumptions $[W]$ of our measurements in which we concentrate on the *Intel Xeon E5-2600v2* generation.



| Intel | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Processor | | | | | | | | | | | Power Consumption @ utilization level [W] | | | | |
| Component | Family | Frequency | | TDP | Max p-state | Hyper-threading cores/threads | Cache | | Transfer rate | | 100 [%] | 80 [%] | 50 [%] | 20 [%] | 0 [%] |
| | Series | Base frequency | Max turbo frequency | | | | L2 | L3 (TLC) | QPI | mem bus | | | | | |
| C1 | E5-2690 | 3.00 GHz | 3.60 GHz | 130 W | 19 | 10C/20T | 10x256KB | 25 MB | 8.0 GT/s | 1866 MHz | 120,5 | 76,83 | 53,62 | 28,38 | 17,61 |
| C2 | E5-2680 | 2.80 GHz | 3.60 GHz | 115 W | 17 | 10C/20T | 10x256KB | 25 MB | 8.0 GT/s | 1866 MHz | 112,88 | 75,16 | 55,84 | 31,07 | 18,85 |
| C3 | E5-2670 | 2.50 GHz | 3.30 GHz | 115 W | 14 | 10C/20T | 10x256KB | 25 MB | 8.0 GT/s | 1866 MHz | 101,22 | 65,38 | 46,66 | 27,99 | 18,13 |
| C4 | E5-2660 | 2.20 GHz | 3.00 GHz | 95 W | 11 | 10C/20T | 10x256KB | 25 MB | 8.0 GT/s | 1866 MHz | 81,2 | 57,07 | 43,76 | 28,41 | 17,01 |
| C5 | E5-2650 | 2.60 GHz | 3.40 GHz | 95 W | 15 | 8C/16T | 8x256KB | 20 MB | 8.0 GT/s | 1866 MHz | 101,1 | 67,08 | 51,8 | 29,12 | 18,45 |
| C6 | E5-2640 | 2.00 GHz | 2.50 GHz | 95 W | 9 | 8C/16T | 8x256KB | 20 MB | 7.2 GT/s | 1600 MHz | 80,2 | 61,76 | 50,16 | 28,04 | 18,93 |
| C7 | E5-2630 | 2.60 GHz | 3.10 GHz | 80 W | 15 | 6C/12T | 6x256KB | 15 MB | 7.2 GT/s | 1866 MHz | 62,69 | 47,31 | 36,83 | 22,44 | 15,44 |
| C8 | E5-2620 | 2.10 GHz | 2.60 GHz | 80 W | 10 | 6C/12T | 6x256KB | 15 MB | 7.2 GT/s | 1600 MHz | 54,07 | 42,41 | 32,55 | 21,02 | 15,41 |
| C9 | E5-2609 | 2.50 GHz | 2.50 GHz | 80 W | 14 | 4C/4T | 4x256KB | 10 MB | 6.4 GT/s | 1333 MHz | 38,96 | 33,31 | 25,69 | 19,2 | 14,68 |
| C10 | E5-2603 | 1.80 GHz | 1.80 GHz | 80 W | 7 | 4C/4T | 4x256KB | 10 MB | 6.4 GT/s | 1333 MHz | 31,23 | 27,23 | 24,17 | 17,85 | 14,69 |
| C11 | E5-2697 | 2.70 GHz | 3.50 GHz | 130 W | 16 | 12C/24T | 12x256KB | 30 MB | 8.0 GT/s | 1866 MHz | 132,21 | 98,56 | 67,8 | 43,7 | 23,63 |
| C12 | E5-2695 | 2.40 GHz | 3.20 GHz | 115 W | 13 | 12C/24T | 12x256KB | 30 MB | 8.0 GT/s | 1866 MHz | 117,93 | 80,57 | 64,09 | 35,22 | 21,67 |
| C13 | E5-2667 | 3.30 GHz | 4.00 GHz | 130 W | 22 | 8C/16T | 8x256KB | 25 MB | 8.0 GT/s | 1866 MHz | 133,95 | 88,9 | 59,76 | 33,72 | 18,17 |
| C14 | E5-2643 | 3.50 GHz | 3.80 GHz | 130 W | 24 | 6C/12T | 6x256KB | 25 MB | 8.0 GT/s | 1866 MHz | 124,57 | 97,66 | 62,2 | 32,46 | 17,57 |
| C15 | E5-2637 | 3.50 GHz | 3.80 GHz | 130 W | 24 | 4C/8T | 4x256KB | 15 MB | 8.0 GT/s | 1866 MHz | 79,31 | 60,89 | 40,46 | 24,51 | 16,32 |
| C16 | E5-2650L | 1.70 GHz | 2.10 GHz | 70 W | 6 | 10C/20T | 8x256KB | 25 MB | 8.0 GT/s | 1600 MHz | 65,42 | 56,11 | 37,55 | 29,06 | 18,74 |
| C17 | E5-2630L | 2.40 GHz | 2.80 GHz | 60 W | 13 | 6C/12T | 6x256KB | 15 MB | 7.2 GT/s | 1600 MHz | 53,32 | 40,39 | 32,9 | 20,49 | 14,42 |
| C18 | E5-2650 | 2.10 GHz | 2.3 GHz | 70 W | 10 | 8C/16T | 8x256KB | 20 MB | 8.0 GT/s | 1600 MHz | 50,58 | 42,77 | 29,74 | 21,85 | 14,12 |
| C19 | E5-2650 | 2.00 GHz | 2.2 GHz | 95 W | 9 | 8C/16T | 8x256KB | 20 MB | 8.0 GT/s | 1600 MHz | 84,3 | 68,7 | 50 | 34,5 | 25,5 |

Figure 167: Processor characteristics overview $(C1 - C19)$ and measurements

We exemplarily describe our results concerning the thermal design power (TDP), p-state, frequency, quick path interconnect (QPI), turbo, or cache. Figure 168 graphically presents the power consumption of the processors at diverse utilization levels. The idle power consumption varies between 14W and 24W, which we define as base power for the *E5-2600v2* family. We include our findings at lower utilization levels to get results that are more precise. The maximal power range differs from 31W up to 134W.

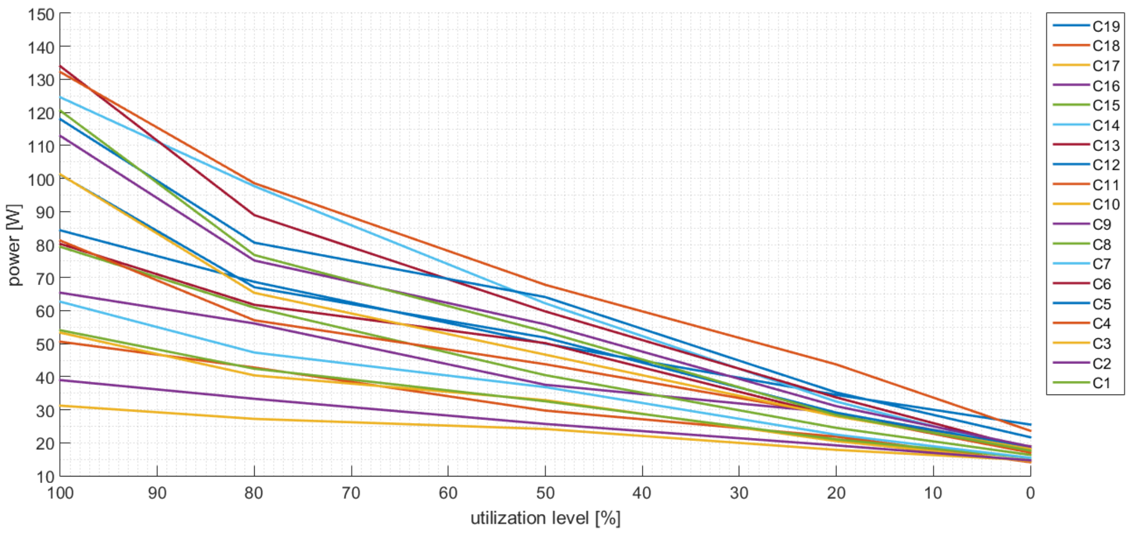


Figure 168: Intel Xeon E5-2600v2 power consumption measurement $(C1 - C19)$

We cannot find a relation between the highest number of p-state and the power consumption, but we can assume that the p-states define the step resolution, which directly correspond to the utilization levels. If a processor has only six p-states, such as $(C16)$, the power model is less precise than a processor with nineteen p-states.

***Thermal Design Power (TDP), Utilization Levels, P-states, and Frequencies***

The vendor provides the thermal design power (TDP), which is accessible to the public within the technical specification of each processor. We estimate the processor power consumption on the basis of the spreadsheet facts. We cannot specify a power curve when we have only the largest power consumption. We need further data regarding the processor power consumption, under ideal conditions for every intermediate utilization level. Usually, the processor vendor roughly specifies the power consumption by idle, average, and full utilization for business and collaborative partners. We received an internal spreadsheet about the processor power consumption at the certain utilization levels, because we cooperated with Fujitsu Technology Solutions GmbH[422]. The processor power consumption is a nearly linear function. The spreadsheet contains the maximal frequency $f_{max}$ at the p-state $P_1$. The vendor has not provided the maximal number of p-states $k$. We analyze the *E5-2600* product family, which has always a minimal frequency of $f_{min} = 1.20GHz$, but we do not know the corresponding p-state. The frequency changes in equidistant steps of $\Delta f = 0.1GHz$, which is specific to the architecture. For instance, the processor $C_1$ has a maximal frequency of $f_{max} = 3.0GHz$. We calculate the number of p-states $k$ by the ratio of frequencies and their step size. We receive a set of available frequencies, Equation (A3.29), based on an Equation (A3.28).

$$k = \frac{f_{max} - f_{min}}{\Delta f} \tag{A3.28}$$

$$f = \{f_{max}, f_{max} - 1 * \Delta f, f_{max} - 2 * \Delta f, \dots, f_{max} - (k-1) * \Delta f, f_{min}\} \tag{A3.29}$$

The thermal design power $TDP = 130W$ is the value of the maximal frequency $f_{max}$. The vendor has not provided the minimal power, but we can estimate the power using the slope of the power curve $\Delta PO_{proc}$ which defines the power reduction of each $\Delta f$. We receive the $\Delta PO_{proc}$ in our cooperation, which we alternatively have to measure. We subtract the $\Delta PO_{proc}$ value of the $TDP$ by the factor of frequency changes $k$. Equation (A3.30) shows our spreadsheet-based estimation.

$$PO_{proc} = \{TDP, TDP - 1 * \Delta PO_{proc}, TDP - 2 * \Delta PO_{proc}, \dots, TDP - k * \Delta PO_{proc}\} \tag{A3.30}$$

[422] Fujitsu Technology Solutions GmbH: http://www.fujitsu.com/fts/

We analyze the processors $C_1 - C_{17}$ and overestimate the power approximately 25% in relative deviation compared to the spreadsheet. Our estimation method underestimates the power nearly 3% in relative deviation. Table 109 provides the slope values and the concrete relative deviations.

Table 109: Intel spreadsheet vs. estimation method

| Component | Family - series | Slope of the power curve $\Delta\mathrm{PO}_{\mathrm{proc}}$ | Relative over-estimation [%] | Relative under-estimation [%] |
|---|---|---|---|---|
| C1 | Intel Xeon E5-2690v2 | 4.06 | 14.86 | 0 |
| C2 | Intel Xeon E5-2680v2 | 3.38 | 8.91 | 0 |
| C3 | Intel Xeon E5-2670v2 | 3.38 | 4.85 | 0.54 |
| C4 | Intel Xeon E5-2660v2 | 2.7 | 0.33 | 2.72 |
| C5 | Intel Xeon E5-2650v2 | 2.5 | 6.25 | 0 |
| C6 | Intel Xeon E5-2640v2 | 2.38 | 0.48 | 1.72 |
| C7 | Intel Xeon E5-2630v2 | 1.79 | 4.52 | 0 |
| C8 | Intel Xeon E5-2620v2 | 1.67 | 1.01 | 0.9 |
| C9 | Intel Xeon E5-2609v2 | 1.23 | 2.02 | 0.45 |
| C10 | Intel Xeon E5-2603v2 | 1.67 | 0.46 | 0.46 |
| C11 | Intel Xeon E5-2697v2 | 4.2 | 5.93 | 0 |
| C12 | Intel Xeon E5-2695v2 | 3.83 | 4.55 | 0.78 |
| C13 | Intel Xeon E5-2667v2 | 3.33 | 18.28 | 0 |
| C14 | Intel Xeon E5-2643v2 | 3.17 | 24.53 | 0 |
| C15 | Intel Xeon E5-2637v2 | 2.74 | 20.1 | 0 |
| C16 | Intel Xeon E5-2650Lv2 | 2.4 | 0.67 | 1.21 |
| C17 | Intel Xeon E5-2630Lv2 | 1.33 | 3.14 | 0 |

Our estimation method becomes inaccurate (more than ten percent) for processors with more than 15 p-states $(C_1, C_{13}, C_{14}, C_{15})$. In this case, we will include a non-linear basis for our power calculation method. The dashed line in Figure 169 presents our estimation values, and the solid line shows the spreadsheet power consumption. We evaluate the average deviation, which is 7% for over-estimation and 1% for under-estimation. We neglect the differences of the processor power consumption and conclude that our estimation[423] method is almost identical with the vendor (industrial tools), which rely upon the spreadsheets.

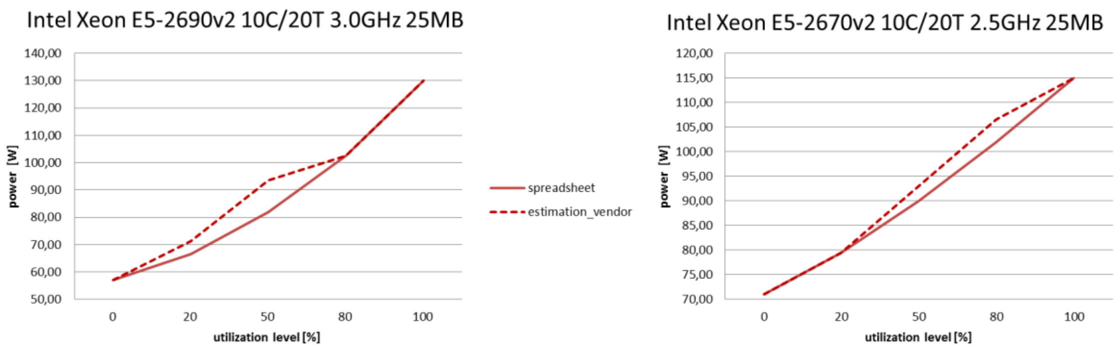


Figure 169: Processor $(C1)$ and $(C3)$ – spreadsheet vs. vendor estimation

Nevertheless, we observe a power gap between our spreadsheet-based estimation and our measurements, shown in Figure 170 and Figure 171. In our example, the measured power at the idle utilization level (~20W) is approximately 30% of the spreadsheet-based estimated power (60-70W). At a utilization level of 50%, the imprecision shrinks and the measured power is nearly 60% of the estimation. Finally, the power gap is less than 10% at a utilization level of 100%. The power gap for the idle utilization is larger compared to the full utilization.

[423] Power estimation: our spreadsheet-based method is synonymous with the vendor estimation.

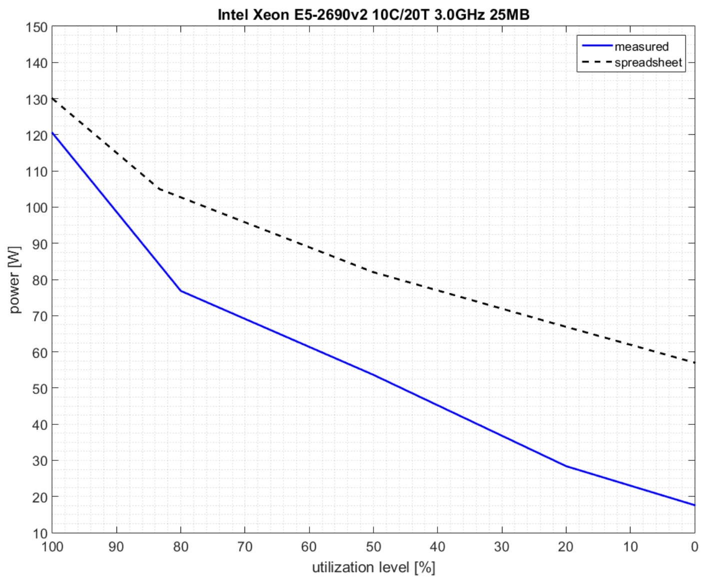


**Figure 170: Spreadsheet-based estimation vs. measurements of the processor $(C1)$**

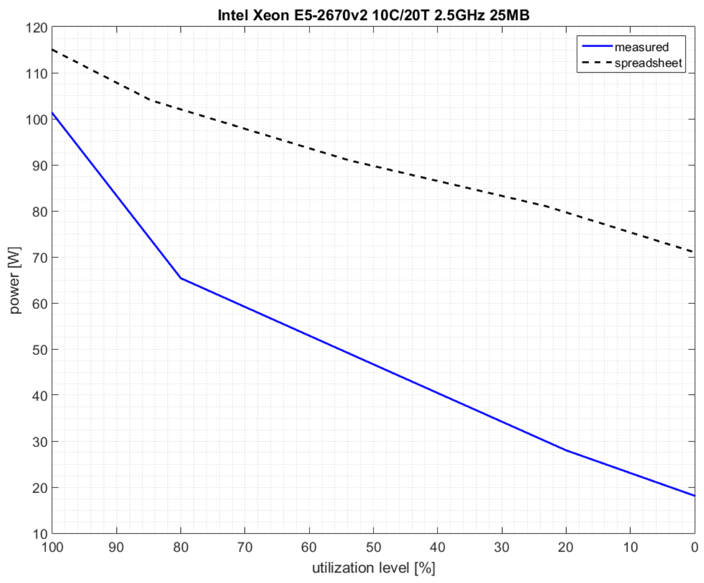


**Figure 171: Spreadsheet-based estimation vs. measurements of the processor $(C3)$**

As a result, we see that the vendor or our spreadsheet-based method[424] overestimates the power consumption for two processors of the product family *E5-2600,* which we pick for our example. We compared other families and saw the same problem. We conclude that the vendor always overestimates the power consumption of the processor product family *E5-2600*.

[424] Vendor, spreadsheet-based method: we receive the power consumption from the industrial tools and compare them with our estimation method to become nearly identical

Our aim is to conceive a non-linear processor power model, which reduces the gap between the spreadsheet-based estimation and the measurements. We reduce the worst-case power assumption and enhance the power specification at lower utilization levels. We define a spreadsheet-based method considering diverse characteristics from the technical specification available to the public. We try to model the power consumption by ten percent in a single utilization step, which refers to the processor states.

### *Quick Path Interconnects (QPI)*

The quick path interconnect (QPI) is a point-to-point interconnect between the processor and the memory controller. At the idle utilization level, the power increases by the QPI transfer rate, as shown in Table 110. Finally, we assume that the idle power consumption depends on the QPI speed.

Table 110: Average power consumption $(C1 - C19)$ at 0% utilization level

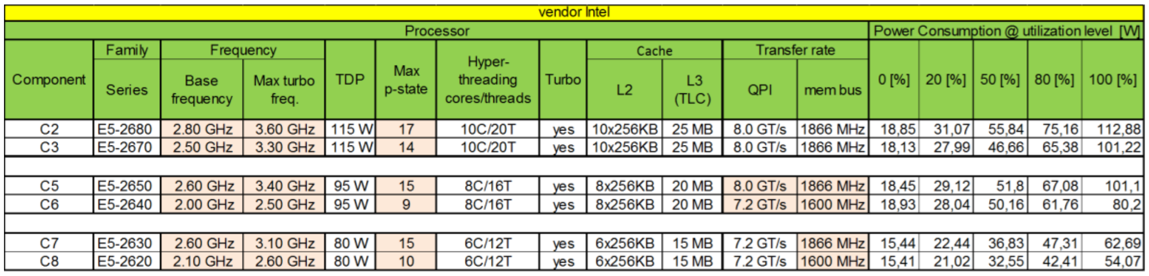
| QPI speed [GT/s] | Average power consumption @ 0% utilization level [W] |
|---|---|
| **6.4** | 14.7 |
| **7.2** | 16.1 |
| **8.0** | 18.7 |

The thermal design power defines the processor power limit at a utilization level of 100%. We analyze the maximal power consumption and propose a TDP-based power function, see Equation (A3.31).

$$power_{Intel\ Xeon\ E5-2600v2}(100\%) = \begin{cases} TDP * 1.04, & if\ L2\ cache \geq 12x256KB, L3\ cache \geq 30MB, threads \geq 24 \\ TDP * 0.54, & if\ L2\ cache \leq\ \ 4x256KB, L3\ cache \leq 15MB, no\ turbo \\ TDP * 0.91, & otherwise \end{cases} \quad \text{(A3.31)}$$

### *Frequency, TDP, Hyper-Threading, Turbo, Cache, and Transfer Rate*

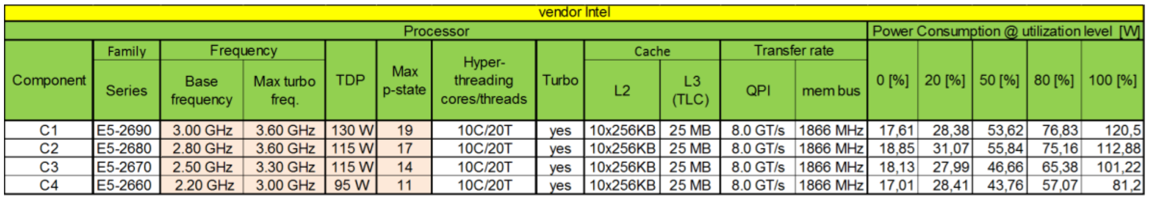
We analyze the component pairs $(C2, C3; C5, C6)$ and $(C7, C8)$, which have the same TDP values to specify a model, which has a finer granularity. Figure 172 summarizes the power consumptions, which show that the TDP and frequency influence grow when the utilization level is higher than 50%.



| vendor Intel | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Processor | | | | | | | | | | | | Power Consumption @ utilization level [W] | | | | |
| Component | Family | Frequency | | TDP | Max p-state | Hyper-threading cores/threads | Turbo | Cache | | Transfer rate | | 0 [%] | 20 [%] | 50 [%] | 80 [%] | 100 [%] |
| | Series | Base frequency | Max turbo freq. | | | | | L2 | L3 (TLC) | QPI | mem bus | | | | | |
| C2 | E5-2680 | 2.80 GHz | 3.60 GHz | 115 W | 17 | 10C/20T | yes | 10x256KB | 25 MB | 8.0 GT/s | 1866 MHz | 18,85 | 31,07 | 55,84 | 75,16 | 112,88 |
| C3 | E5-2670 | 2.50 GHz | 3.30 GHz | 115 W | 14 | 10C/20T | yes | 10x256KB | 25 MB | 8.0 GT/s | 1866 MHz | 18,13 | 27,99 | 46,66 | 65,38 | 101,22 |
| | | | | | | | | | | | | | | | | |
| C5 | E5-2650 | 2.60 GHz | 3.40 GHz | 95 W | 15 | 8C/16T | yes | 8x256KB | 20 MB | 8.0 GT/s | 1866 MHz | 18,45 | 29,12 | 51,8 | 67,08 | 101,1 |
| C6 | E5-2640 | 2.00 GHz | 2.50 GHz | 95 W | 9 | 8C/16T | yes | 8x256KB | 20 MB | 7.2 GT/s | 1600 MHz | 18,93 | 28,04 | 50,16 | 61,76 | 80,2 |
| | | | | | | | | | | | | | | | | |
| C7 | E5-2630 | 2.60 GHz | 3.10 GHz | 80 W | 15 | 6C/12T | yes | 6x256KB | 15 MB | 7.2 GT/s | 1866 MHz | 15,44 | 22,44 | 36,83 | 47,31 | 62,69 |
| C8 | E5-2620 | 2.10 GHz | 2.60 GHz | 80 W | 10 | 6C/12T | yes | 6x256KB | 15 MB | 7.2 GT/s | 1600 MHz | 15,41 | 21,02 | 32,55 | 42,41 | 54,07 |

Figure 172: Processor frequency, p-state, and transfer rate $(C2, C3, C5 - C8)$

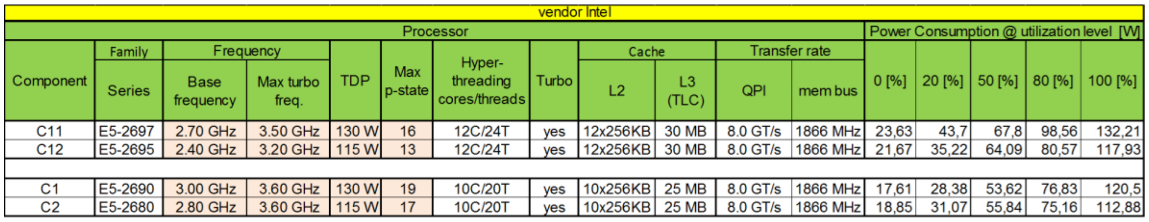
Figure 173 contains processors that all have the same amount of cores/threads, an identical cache size, transfer rate, and support the turbo mode. The processors $(C1 - C4)$ differ in their base frequency, TDP, and maximum p-states. The power consumption fluctuates up to 10W at a utilization level of 0%, 20%, and 50%. The processors $(C2)$ and $(C3)$ have the same TDP, but a $0.3GHz$ variation in their base frequency as well as maximum frequency. At a utilization level of 80% and 100%, the power consumption increases with a higher frequency. A small base frequency and TDP value, see $(C4)$, causes less power consumption at an 80% or 100% utilization level compared to a higher frequency-TDP pair $(C1)$.



| vendor Intel | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Processor | | | | | | | | | | | | Power Consumption @ utilization level [W] | | | | |
| Component | Family | Frequency | | TDP | Max p-state | Hyper-threading cores/threads | Turbo | Cache | | Transfer rate | | 0 [%] | 20 [%] | 50 [%] | 80 [%] | 100 [%] |
| | Series | Base frequency | Max turbo freq. | | | | | L2 | L3 (TLC) | QPI | mem bus | | | | | |
| C1 | E5-2690 | 3.00 GHz | 3.60 GHz | 130 W | 19 | 10C/20T | yes | 10x256KB | 25 MB | 8.0 GT/s | 1866 MHz | 17,61 | 28,38 | 53,62 | 76,83 | 120,5 |
| C2 | E5-2680 | 2.80 GHz | 3.60 GHz | 115 W | 17 | 10C/20T | yes | 10x256KB | 25 MB | 8.0 GT/s | 1866 MHz | 18,85 | 31,07 | 55,84 | 75,16 | 112,88 |
| C3 | E5-2670 | 2.50 GHz | 3.30 GHz | 115 W | 14 | 10C/20T | yes | 10x256KB | 25 MB | 8.0 GT/s | 1866 MHz | 18,13 | 27,99 | 46,66 | 65,38 | 101,22 |
| C4 | E5-2660 | 2.20 GHz | 3.00 GHz | 95 W | 11 | 10C/20T | yes | 10x256KB | 25 MB | 8.0 GT/s | 1866 MHz | 17,01 | 28,41 | 43,76 | 57,07 | 81,2 |

Figure 173: Processor frequency, TDP, and p-state $(C1 - C4)$

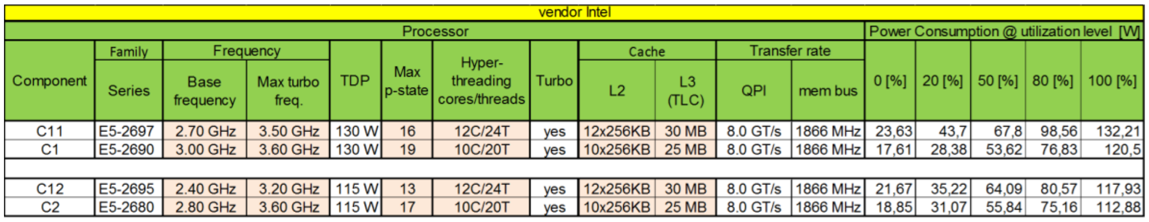
Figure 174 and Figure 175 present processor pairs that are consistent in their cores/threads and cache size. We analyze the same processors, but compare different components that have the same TDP value and transfer rate. We analyze the pair $(C11, C12)$ and $(C1, C2)$, which have a difference of 15W in the TDP. We assume the power consumption behaves at the full utilization level in a similar way.



| vendor Intel | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Processor | | | | | | | | | | | | Power Consumption @ utilization level [W] | | | | |
| Component | Family | Frequency | | TDP | Max p-state | Hyper-threading cores/threads | Turbo | Cache | | Transfer rate | | 0 [%] | 20 [%] | 50 [%] | 80 [%] | 100 [%] |
| | Series | Base frequency | Max turbo freq. | | | | | L2 | L3 (TLC) | QPI | mem bus | | | | | |
| C11 | E5-2697 | 2.70 GHz | 3.50 GHz | 130 W | 16 | 12C/24T | yes | 12x256KB | 30 MB | 8.0 GT/s | 1866 MHz | 23,63 | 43,7 | 67,8 | 98,56 | 132,21 |
| C12 | E5-2695 | 2.40 GHz | 3.20 GHz | 115 W | 13 | 12C/24T | yes | 12x256KB | 30 MB | 8.0 GT/s | 1866 MHz | 21,67 | 35,22 | 64,09 | 80,57 | 117,93 |
| | | | | | | | | | | | | | | | | |
| C1 | E5-2690 | 3.00 GHz | 3.60 GHz | 130 W | 19 | 10C/20T | yes | 10x256KB | 25 MB | 8.0 GT/s | 1866 MHz | 17,61 | 28,38 | 53,62 | 76,83 | 120,5 |
| C2 | E5-2680 | 2.80 GHz | 3.60 GHz | 115 W | 17 | 10C/20T | yes | 10x256KB | 25 MB | 8.0 GT/s | 1866 MHz | 18,85 | 31,07 | 55,84 | 75,16 | 112,88 |

Figure 174: Processor frequency, TDP, and p-state $(C1, C2, C11, C12)$

In addition, we compare $(C11, C1)$ and $(C12, C2)$, whereby the frequency increases, but the L2 and L3 cache decreases. The processors of $(C11)$ and $(C12)$ have two more physical cores and four threads compared to $(C1)$ and $(C2)$. The power ratio $(C11, C1)$ is proportional to $(C12, C2)$, whereby the base frequency is negligible.



| vendor Intel | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Processor | | | | | | | | | | | | | Power Consumption @ utilization level [W] | | | | |
| | Family | Frequency | | | | | | | Cache | | Transfer rate | | | | | | |
| Component | Series | Base frequency | Max turbo freq. | TDP | Max p-state | Hyper-threading cores/threads | Turbo | L2 | L3 (TLC) | QPI | mem bus | 0 [%] | 20 [%] | 50 [%] | 80 [%] | 100 [%] |
| C11 | E5-2697 | 2.70 GHz | 3.50 GHz | 130 W | 16 | 12C/24T | yes | 12x256KB | 30 MB | 8.0 GT/s | 1866 MHz | 23,63 | 43,7 | 67,8 | 98,56 | 132,21 |
| C1 | E5-2690 | 3.00 GHz | 3.60 GHz | 130 W | 19 | 10C/20T | yes | 10x256KB | 25 MB | 8.0 GT/s | 1866 MHz | 17,61 | 28,38 | 53,62 | 76,83 | 120,5 |
| | | | | | | | | | | | | | | | | |
| C12 | E5-2695 | 2.40 GHz | 3.20 GHz | 115 W | 13 | 12C/24T | yes | 12x256KB | 30 MB | 8.0 GT/s | 1866 MHz | 21,67 | 35,22 | 64,09 | 80,57 | 117,93 |
| C2 | E5-2680 | 2.80 GHz | 3.60 GHz | 115 W | 17 | 10C/20T | yes | 10x256KB | 25 MB | 8.0 GT/s | 1866 MHz | 18,85 | 31,07 | 55,84 | 75,16 | 112,88 |

Figure 175: Processor frequency, p-state, and cache $(C1, C2, C11, C12)$

### *Processor Core Frequencies* $(C18)$

Table 111 and Table 112 present thread-specific utilization levels of the processor in relation to the target throughput level in which we found the irrelevance of the specific thread when we observe the entire processor utilization.

Table 111: Thread-specific (#) utilization levels [%] of the processor $(C18)$ as mean values at target throughput $(calibration, 100\% - 60\%)$ in $(SP1.2.8)$

| | Target throughput [%] | | | | | |
|---|---|---|---|---|---|---|
| Processor threads (#) and utilization levels in [%] | Calibration | 100 | 90 | 80 | 70 | 60 |
| **0** | 99.44 | **86.77** | **64.67** | **58.03** | **53.20** | **49.61** |
| **1** | 99.41 | 94.08 | 76.56 | 67.59 | 62.50 | 55.06 |
| **2** | 99.37 | 95.47 | 82.39 | 72.84 | 66.96 | 60.26 |
| **3** | 99.46 | 96.71 | 83.44 | 74.05 | 64.75 | 55.60 |
| ***4*** | 98.97 | 92.28 | 77.94 | 70.42 | 64.84 | 59.34 |
| ***5*** | 99.43 | 96.83 | 83.41 | 73.39 | 63.38 | 53.69 |
| **6** | 99.14 | 95.85 | 84.50 | 76.21 | 68.59 | 58.62 |
| **7** | 99.11 | 94.03 | 76.16 | 65.24 | 57.42 | 50.70 |
| **8** | 99.54 | 96.89 | 83.81 | 75.27 | 65.76 | 59.63 |
| **9** | 99.51 | 96.42 | 84.89 | 76.55 | 66.09 | 54.58 |
| **10** | 99.54 | 97.14 | 86.14 | 77.23 | 67.84 | 58.75 |
| **11** | 99.50 | 96.53 | 84.01 | 74.44 | 63.68 | 54.68 |
| **12** | 99.52 | 94.25 | 79.82 | 71.16 | 65.77 | 59.25 |
| **13** | 99.50 | 97.12 | 85.13 | 76.30 | 64.22 | 53.97 |
| **14** | 99.46 | 92.98 | 78.36 | 70.10 | 64.17 | 58.45 |
| **15** | 99.48 | 97.08 | 86.02 | 75.94 | 65.43 | 55.15 |
| | | | | | | |
| **Mean of all threads (*HW Monitor Pro*)** | **99.4** | **95.03** | **81.08** | **72.17** | **64.04** | **56.08** |
| | | | | | | |
| **Mean processor (*Intel Power Thermal Utility)*** | **99.73** | **96.12** | **84.17** | **75.89** | **67.94** | **60.17** |

**Table 112: Thread-specific (#) utilization levels [%] of the processor $(C18)$ as mean values at target throughput $(50\% - 0\%, idle)$ in $(SP1.2.8)$**

| Processor threads (#) and utilization levels in [%] | Target throughput [%] | | | | | |
|---|---|---|---|---|---|---|
| | 50 | 40 | 30 | 20 | 10 | Idle |
| **0** | **44.07** | **38.67** | **35.31** | **31.02** | **28.64** | **34.4** |
| **1** | 43.58 | 32.95 | 19.60 | 12.77 | 5.25 | 3.1 |
| **2** | 54.15 | 42.50 | 34.91 | 22.35 | 10.92 | **23.4** |
| **3** | 42.28 | 32.38 | 21.07 | 15.13 | 6.64 | 12.5 |
| ***4*** | 54.69 | 43.55 | 34.68 | 22.19 | 11.61 | 3.1 |
| ***5*** | 40.83 | 31.11 | 20.46 | 13.76 | 5.78 | 12.3 |
| **6** | 53.01 | 42.81 | 34.59 | 21.78 | 9.50 | 4.6 |
| **7** | 40.10 | 30.43 | 20.63 | 14.47 | 7.80 | 0 |
| **8** | 50.92 | 34.00 | 32.63 | 22.25 | 12.70 | 10.8 |
| **9** | 42.41 | 32.25 | 22.73 | 14.85 | 8.06 | 7.8 |
| **10** | 51.50 | 41.58 | 35.56 | 24.10 | 13.56 | **23.4** |
| **11** | 42.15 | 30.19 | 19.21 | 13.42 | 6.43 | 15.2 |
| **12** | 54.53 | 46.21 | 36.18 | 26.20 | 13.38 | 7.7 |
| **13** | 40.57 | 30.55 | 21.60 | 13.81 | 9.47 | 10.8 |
| **14** | 53.34 | 42.88 | 35.00 | 23.00 | 11.86 | 7.8 |
| **15** | 42.45 | 31.14 | 19.58 | 14.13 | 6.23 | 10.8 |
| | | | | | | |
| **Mean of all threads (*HW Monitor Pro*)** | **46.91** | **36.83** | **27.73** | **19.08** | **10.49** | **11.73** |
| | | | | | | |
| **Mean processor (*Intel Power Thermal Utility*)** | **50.75** | **40.75** | **31.57** | **22.38** | **13.40** | **4.4** |

## A3g. Detailed Processor Characteristics $(C1 - C4, C7, C8, C14, C18)$

In this section, we present the processor characteristics that we use in our simulation and evaluation.

Table 113: Detailed processor characteristics $(C1, C3, C7)$

| Processor characteristics | Processor $(C1)$ | Processor $(C3)$ | Processor $(C7)$ |
|---|---|---|---|
| L3 cache [MB] | {'25MB'} | {'25MB'} | {'15MB'} |
| L2 cache [KB] | {'2056KB'} | {'2056KB}' | {'1536KB'} |
| Semiconductor technology (TDP) [W] | {'130W'} | {'115W'} | {'80W'} |
| Vendor | {'Intel'} | {'Intel'} | {'Intel'} |
| Architecture | Intel *XEON E5* | Intel *XEON E5* | Intel *XEON E5* |
| Generation | Ivy Bridge EP | Ivy Bridge EP | Ivy Bridge EP |
| Family | *E5-2600v2* | *E5-2600v2* | *E5-2600v2* |
| Series | {'E5-2690v2'} | {'E5-2670v2'} | {'E5-2630v2'} |
| Cores / active cores (hyper-threading) | {'10C'}, {'20T'} | {'10C'}, {'20T'} | {'6C'}, {'12T'} |
| Frequency [GHz] | {'3.00'} | {'2.50'} | {'2.60'} |
| Transfer rate [GT/s, MHz] | {'8.0GT/s'}, {'1866MHz'} | {'8.0GT/s'}, {'1866MHz'} | {'7.2GT/s'}, {'1866MHz'} |

Table 114: Detailed processor characteristics $(C2, C4, C8)$

| Processor characteristics | Processor $(C2)$ | Processor $(C4)$ | Processor $(C8)$ |
|---|---|---|---|
| L3 cache [MB] | {'25MB'} | {'25MB'} | {'15MB'} |
| L2 cache [KB] | {'2056KB'} | {'2056KB'} | {'1536KB'} |
| Semiconductor technology (TDP) [W] | {'115W'} | {'95W'} | {'80W'} |
| Vendor | {'Intel'} | {'Intel'} | {'Intel'} |
| Architecture | Intel *XEON E5* | Intel *XEON E5* | Intel *XEON E5* |
| Generation | Ivy Bridge EP | Ivy Bridge EP | Ivy Bridge EP |
| Family | *E5-2600v2* | *E5-2600v2* | *E5-2600v2* |
| Series | {'E5-2680v2'} | {'E5-2660v2'} | {'E5-2620v2'} |
| Cores / active cores (hyper-threading) | {'10C'}, {'20T'} | {'10C'}, {'20T'} | {'6C'}, {'12T'} |
| Frequency [GHz] | {'2.80'} | {'2.20'} | {'2.10'} |
| Transfer rate [GT/s, MHz] | {'8.0GT/s'}, {'1866MHz'} | {'8.0GT/s'}, {'1866MHz'} | {'7.2GT/s'}, {'1600MHz'} |

Table 115: Detailed processor characteristics $(C14, C18)$

| Processor characteristics | Processor $(C14)$ | Processor $(C18)$ |
|---|---|---|
| L3 cache [MB] | `{'25MB'}` | `{'20MB'}` |
| L2 cache [KB] | `{'1536KB'}` | `{'2048KB'}` |
| Semiconductor technology (TDP) [W] | `{'130W'}` | `{'70W'}` |
| Vendor | `{'Intel'}` | `{'Intel'}` |
| Architecture | Intel *XEON E5* | Intel *XEON E5* |
| Generation | Ivy Bridge EP | Ivy Bridge EP |
| Family | *E5-2600v2* | *E5-2600v2* |
| Series | `{'E5-2643v2'}` | `{'E5-2650v2'}` |
| Cores / active cores (hyper-threading) | `{'6C'}`, `{'12T'}` | `{'8C'}`, `{'16T'}` |
| Frequency [GHz] | `{'3.50'}` | `{'2.1'}`, turbo `{'2.3'}` |
| Transfer rate [GT/s, MHz] | `{'8.0GT/s'}`, `{'1866MHz'}` | `{'8.0GT/s'}`, `{'1600MHz'}` |

## A3h. Evaluation Results of the Processors $(C1, C3, C7)$

Figure 176, Figure 177, and Figure 178 show the simulation-based results of the processor power (dash-dotted red line) in comparison to the measurement trace of the *Intel Power Thermal Utility* (solid blue line), the data gained from the commercial tools (dotted magenta line), or vendor-based data determined in the spreadsheets (dashed black line). These figures present the power comparisons of the processors $(C1)$, $(C3)$, and $(C7)$.

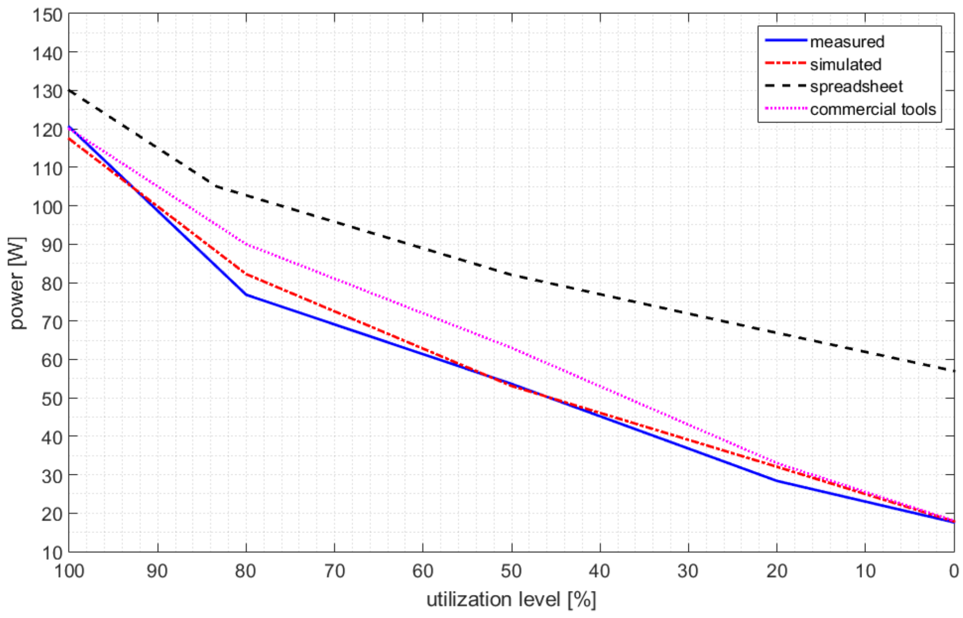


**Figure 176: Processor power consumption $(C1)$ –
an exemplary comparison between spreadsheet, commercial tools, and measured and simulated power**

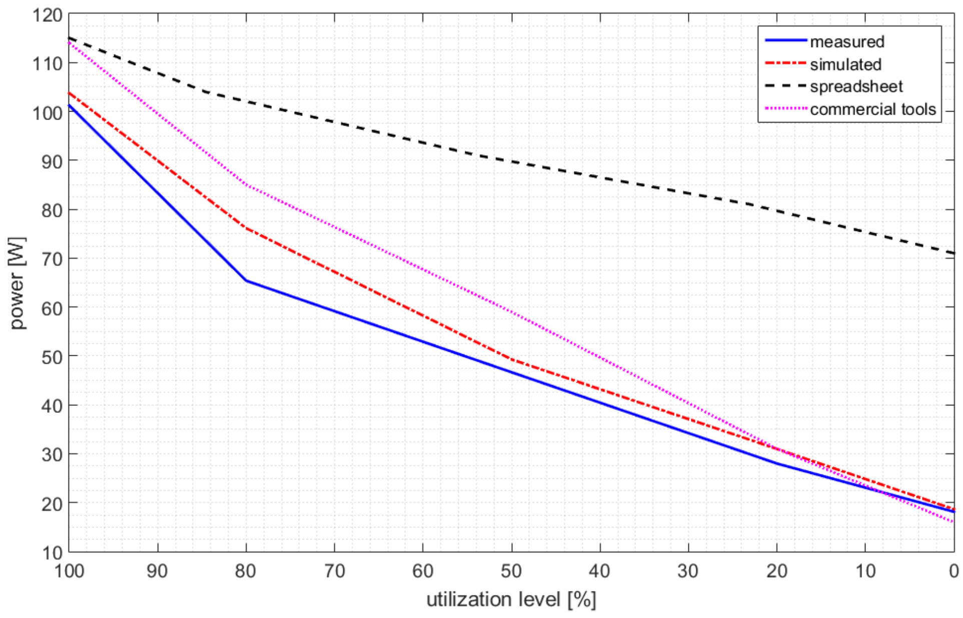


**Figure 177: Processor power consumption $(C3)$ –
an exemplary comparison between spreadsheet, commercial tools, and measured and simulated power**

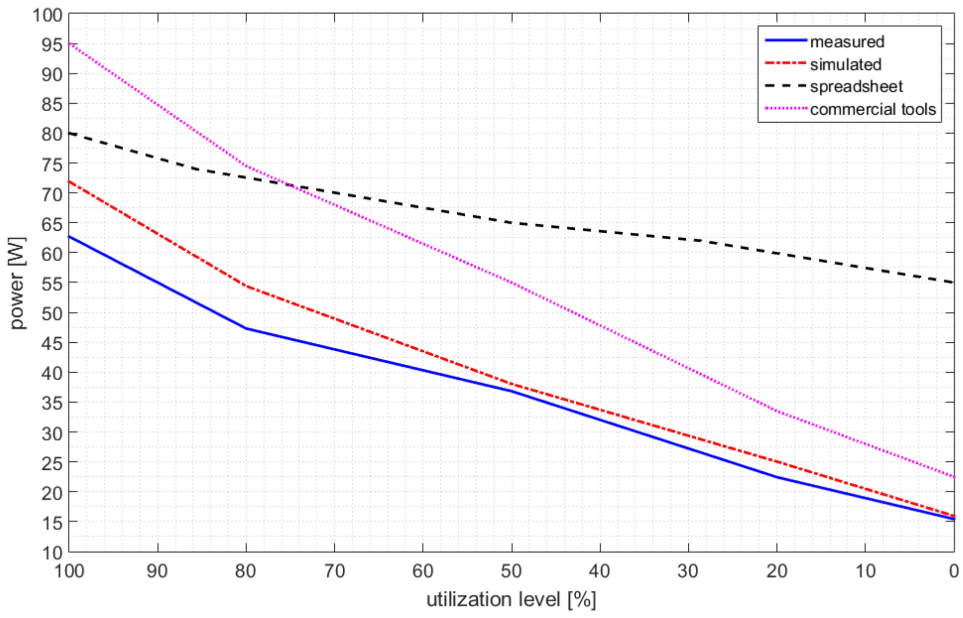


**Figure 178: Processor power consumption $(C7)$ –
an exemplary comparison between spreadsheet, commercial tools, and measured and simulated power**

Figure 179 presents the simulated power consumption in $[W]$ of the processors $C1 - C19$ at the specific utilization level in $[\%]$ that we gained while executing the *SPECpower* benchmark $(SP1.2.8)$.

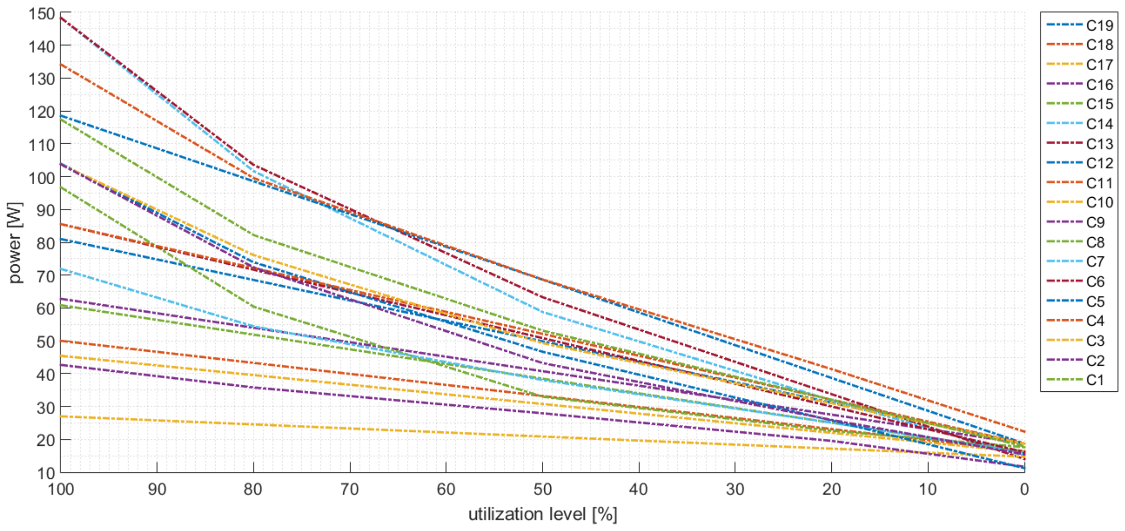


**Figure 179: Simulated processor power consumption $(C1 - C19)$**

## A3i. Memory, Processor, and System Performance Scores

In parallel to the memory and processor evaluation (Section 7.2 and Section 7.3), the particular benchmarks store the highest performance score that we consider when we simulate the energy efficiency, see Section 7.1.3. Figure 180 and Figure 181 exemplarily show the absolute performance scores[425] (on top of the bars) as the results of the *PassMark CPU* $(PCx.y.z)$, the three bars on the left, and the *PassMark Memory* $(PMx.y.z)$ benchmarks the three bars on the right of the figures. It can be observed that if we double the memory capacity in a server system with an exclusive processor $(x = 1)$, either in a single memory module $(\hat{z} = z * 2)$ or as an additional memory module $(\hat{y} = y * 2)$, the performance scores will approximately increase at $(PC1.y.z)$ by $[+4.7, +13.1]\%$ and at $(PM1.y.z)$ by $[+21.7, +28.5]\%$, see Figure 144. If the system has two processors, the impact of the doubled memory capacity increases at the *PassMark CPU* benchmark $(PC2.y.z)$ by $[+6.4, +20.7]$ and increases at the *PassMark Memory* benchmark $(PM2.y.z)$ by $[+5.7, +12.4]\%$, as shown in Figure 181.

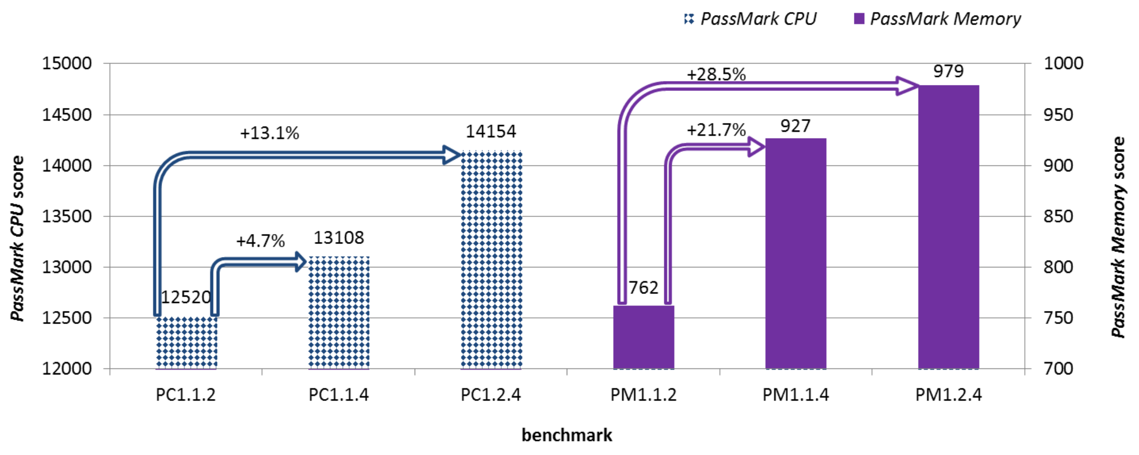


Figure 180: *PassMark CPU* $(PC1.y.z)$ and *PassMark Memory* $(PM1.y.z)$ performance scores

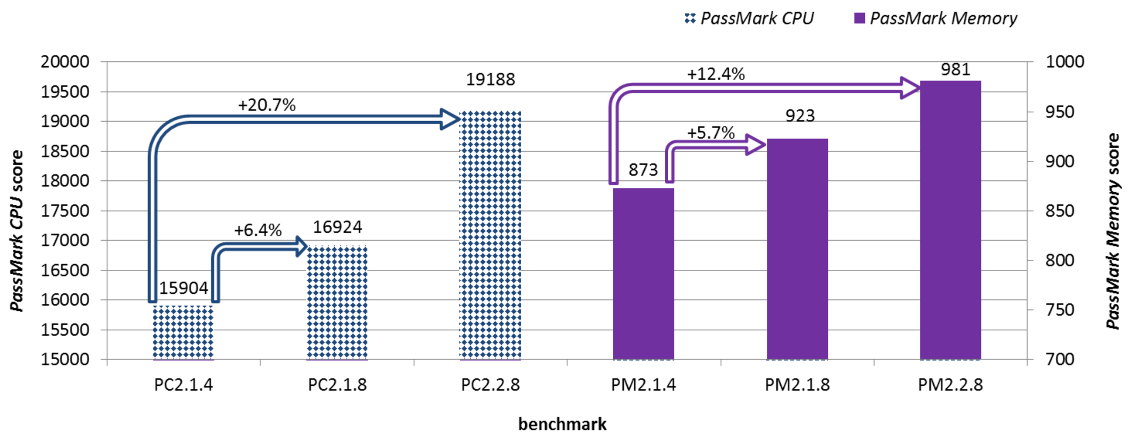


Figure 181: *PassMark CPU* $(PC2.y.z)$ and *PassMark Memory* $(PM2.y.z)$ performance scores

[425] Performance scores: additional scores are available at https://www.memorybenchmark.net/ram_list.php and https://www.cpubenchmark.net/cpu_list.php

Furthermore, we observe that an additional processor and its related second memory module[426] have a significant impact of approximately $14.6\%$ on the performance scores only when each processor has $2GB$ memory capacity, as shown in the *PassMark Memory* benchmarks $(PM1.1.2, PM2.1.4)$, see Figure 182. In our measurements, the performance scores are approximately identical when adding an extra processor. In contrast, when the server system executes the *PassMark CPU* $(PCx.y.z)$ benchmarks in which we provide an additional processor and memory module, we observe a higher performance improvement of $27\%$, $29.1\%$, and $35.6\%$ in comparison to the memory-bounded workload, as shown in Figure 183.

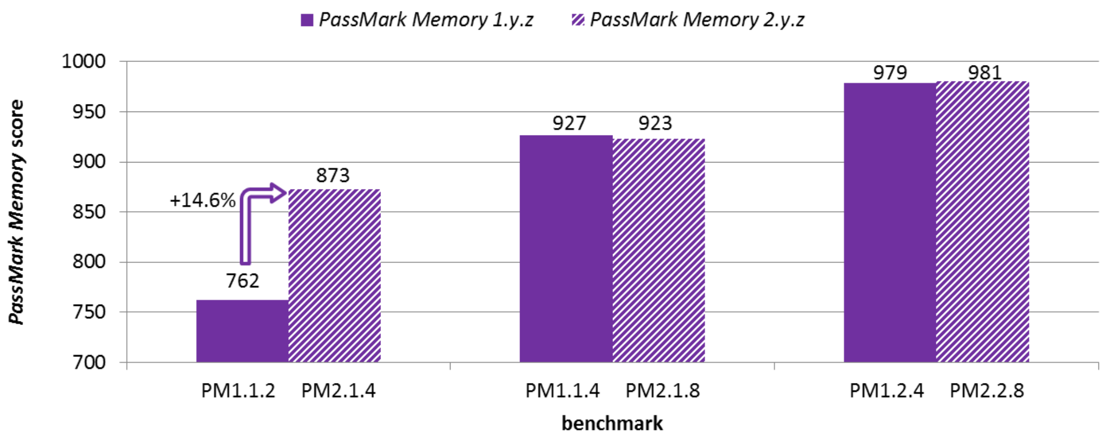


Figure 182: *PassMark Memory* performance scores $(\hat{x} = x * 2)$

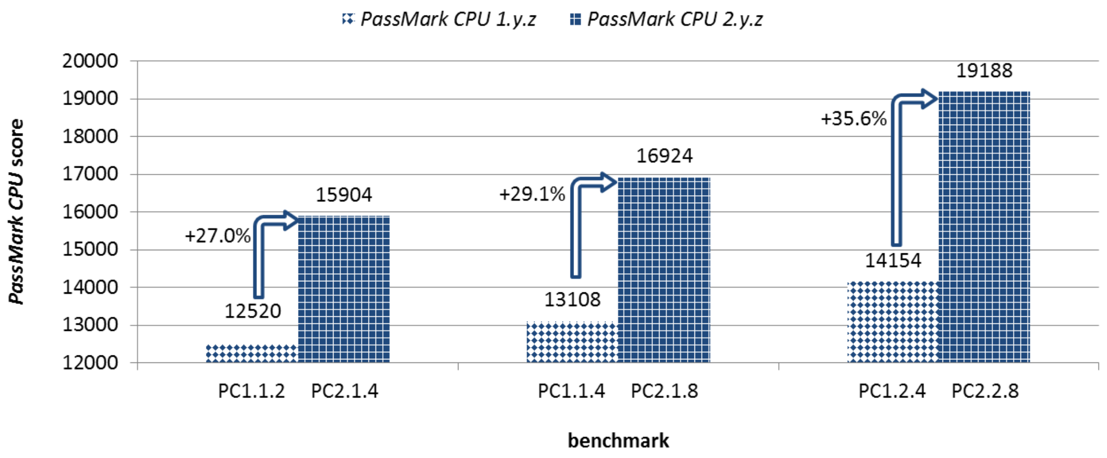
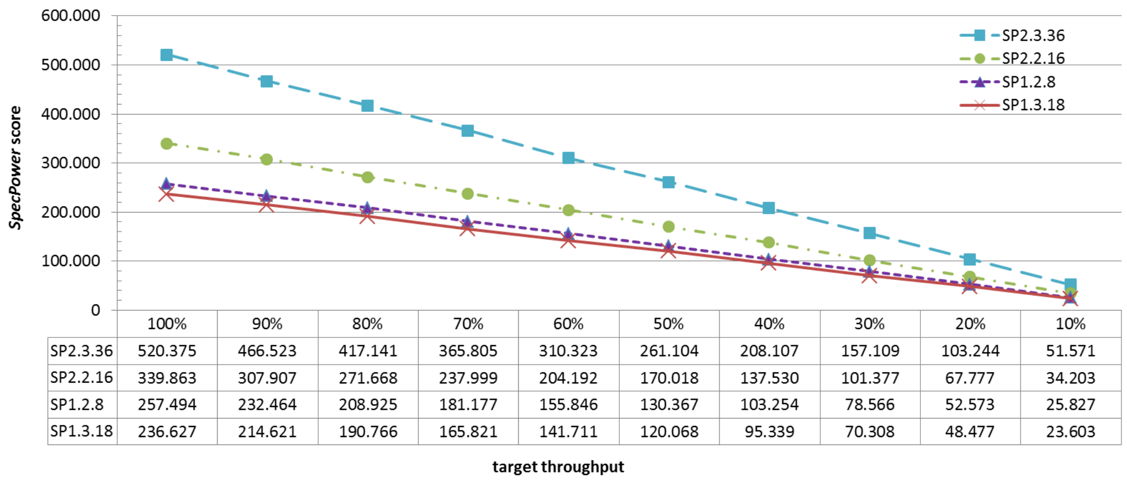


Figure 183: *PassMark CPU* performance scores $(\hat{x} = x * 2)$

Furthermore, we observe linear curves of the performance scores at the specific target throughputs when executing the *SPECpower* benchmarks. Figure 184 exemplarily presents the absolute performance scores considering our system under test of a single processor $(x = 1)$

[426] Second memory module: server system settings require a memory module per processor, based upon the fact of the regular expansion

and two processors ($x = 2$) in ($SPx.y.z$). We observe that an extra processor always increases the peak performance scores. In our example, the performance scores increase by the approximate factor of $1.31$ between ($SP1.2.8$) and ($SP2.2.16$), but the extra processor is more efficient between ($SP1.3.18$) and ($SP2.3.36$) by a factor of nearly 2.2. To our surprise, we observe that the performance scores of the ($SP1.2.8$) are higher in comparison to ($SP1.3.18$), which has more memory capacity ($z = 18$). One reason may be the restricted memory bandwidth or processor cache size. In the case of the configuration, including a second processor we observe that the additional memory capacity significantly improves the performance from $339.863$ at ($SP2.2.16$) up to $520.375$ at ($SP2.3.36$).



| | 100% | 90% | 80% | 70% | 60% | 50% | 40% | 30% | 20% | 10% |
|---|---|---|---|---|---|---|---|---|---|---|
| SP2.3.36 | 520.375 | 466.523 | 417.141 | 365.805 | 310.323 | 261.104 | 208.107 | 157.109 | 103.244 | 51.571 |
| SP2.2.16 | 339.863 | 307.907 | 271.668 | 237.999 | 204.192 | 170.018 | 137.530 | 101.377 | 67.777 | 34.203 |
| SP1.2.8 | 257.494 | 232.464 | 208.925 | 181.177 | 155.846 | 130.367 | 103.254 | 78.566 | 52.573 | 25.827 |
| SP1.3.18 | 236.627 | 214.621 | 190.766 | 165.821 | 141.711 | 120.068 | 95.339 | 70.308 | 48.477 | 23.603 |

**Figure 184: *SPECpower* ($SPx.y.z$) performance scores**

In our simulation model, we create a database to specify the particular component-based as well as system-specific performance scores concerning their various characteristics and benchmarks. Herein, we support a wide variety of hardware and software configurations and their related performance scores.

## A4. List of Tables

## A5. List of Figures

## A6. Bibliography

[Abr 2005]
A. Abraham "Evolutionary multiobjective optimization". Springer London. 1-85233-787-7, 2005.

[AC 2003]
N. Anuar, N. Chiang "Thermal characterization and electrical performance of low profile power packages" Electronics Packaging Technology, 2003 5th Conference (EPTC 2003). 10.1109/EPTC.2003.1271511 Print ISBN:0-7803-8205-6, 2003.

[Acc 2008]
accenture "Data Center Energy Forecast”, 2008.

[ADK et al. 2012]
W. Abdelmaksoud, T. Dang, H. Khalifa, R. Schmidt, M. Iyengar "Perforated tile models for improving data center CFD simulation" Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), 2012 13th IEEE Intersociety Conference on. 10.1109/ITHERM.2012.6231414 Print ISBN:978-1-4244-9533-7, 2012.

[AK 2002]
C. Alexopoulos, S.-H. Kim "OUTPUT DATA ANALYSIS FOR SIMULATIONS" Proceedings of the 2002 Winter Simulation Conference, 2002.

[ALE 2002]
T. Austin, E. Larson, D. Ernst "SimpleScalar: an infrastructure for computer system modeling" IEEE Computer, Heft 2. 10.1109/2.982917 ISSN :0018-9162, 2002.

[AMD 2000]
AMD "AMD PowerNow! Technology Informational White Paper" Advanced Micro Devices, Inc. Revision A, 2000.

[And 2009]
H. Andersson "Aircraft Systems Modeling" ISBN 978-91-7393-692-7, 2009.

[And 2012]

H. Andersson "Variability and Customization of Simulator Products" Linköping Studies in Science and Technology ISBN 978-91-7519-963-4, 2012.

[AP 2014]

R. Alshahrani, H. Peyravi "Modeling and Simulation of Data Center Networks" Proceedings of the 2Nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation. 10.1145/2601381.2601389 ISBN: 978-1-4503-2794-7, 2014.

[APL et al. 2008]

Amit Kumar, P. Princeton Univ., Li Shang, Li-Shiuan Peh, Niraj K. Jha "System-Level Dynamic Thermal Management for High-Performance Microprocessors" Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on. 27, Heft 1. 10.1109/TCAD.2007.907062 ISSN 0278-0070, 2008.

[AR 2016]

I. Ahmad, S. Ranka "Handbook of Energy-Aware and Green Computing" - Two Volume Set. CRC Press. 9781482254440, 2016.

[ASS et al. 2014]

S. Alkharabsheh, B. Sammakia, S. Shrivastava, R. Schmidt "Implementing rack thermal capacity in a room level CFD model of a data center" Semiconductor Thermal Measurement and Management Symposium (SEMI-THERM), 2014 30th Annual. 10.1109/SEMI-THERM.2014.6892237, 2014.

[BBB 2011]

F. Beneventi, A. Bartolini, L. Benini "Static Thermal Model Learning for High-Performance Multicore Servers" Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on. 10.1109/ICCCN.2011.6006065 Print ISBN:978-1-4577-0637-0, 2011.

[BBJ et al. 2009]

J. Bean, R. Bednar, R. Jones, R. Jones, P. Morris, D. Moss, M. Patterson, Prisco, Joe, Vinson, Wade, J. Wallerich "Proper Sizing of IT Power and Cooling Loads" The Green Grid, 2009.

[BBT et al. 2014]

F. Beneventi, A. Bartolini, A. Tilli, L. Benini "An Effective Gray-Box Identification Procedure for Multicore Thermal Modeling" Computers, IEEE Transactions on, Heft 5. 10.1109/TC.2012.293 ISSN :0018-9340, 2014.

[BC 2010]

A. Bohra, V. Chaudhary "VMeter: Power modelling for virtualized clouds" Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on. 10.1109/IPDPSW.2010.5470907 Print ISBN:978-1-4244-6533-0, 2010.

[BCC et al. 2014]

Baker Loyd, P. Clemente, B. Cohen, L. Permenter, B. Purves, P. Salmon "Model Driven System Design Working Group: Foundational Concepts for Model Driven System Design" INCOSE International Symposium. 6. 10.1002/j.2334-5837.1996.tb02139.x, 2014.

[BCS et al. 2012]

P. Bernardi, M. de Carvalho, E. Sanchez, M. Reorda, A. Bosio, L. Dilillo, P. Girard, M. Valka "Peak Power Estimation: A Case Study on CPU Cores". 10.1109/ATS.2012.58 ISSN: 1081-7735, 2012.

[BDM et al. 2008]

J. Branke, K. Deb, K. Miettinen, R. Slowinski "Multiobjective Optimization". 5252. 10.1007/978-3-540-88908-3 ISBN 978-3-540-88907-6, 2008.

[BDW et al. 2007]

Bruno Diniz, Dorgival Guedes, Wagner Meira, Ricardo Bianchini "Limiting the Power Consumption of Main Memory" ISCA '07 Proceedings of the 34th annual international symposium on Computer architecture. 34. 10.1145/1273440.1250699, 2007.

[Bec 2012]

Beckett John "Memory Performance Guidelines for Dell PowerEdge 12th Generation Servers" Enterprise Solutions Group, 2012.

[Bel 2000]

F. Bellosa "The Benefits of Event: Driven Energy Accounting in Power-sensitive Systems" Proceedings of the 9th Workshop on ACM SIGOPS European Workshop: Beyond the PC: New Challenges for the Operating System. 10.1145/566726.566736, 2000.

[Bel 2001]

F. Bellosa "The Case for Event-Driven Energy" Technical Report TR-I4-01-07, Friedrich-Alexander-University Erlangen-Nürnberg, Department of Computer, 2001.

[Ben 2002]

J. Benson "TB379: Thermal Characterization of Packages for ICs", 2002.

[Ben 2010]

L. Benini "Low-Power Integrated Systems" DEIS Doctoral School 2010, 2010.

[BGM et al. 2010]

R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, E. Ayguade "Decomposable and Responsive Power Models for Multicore Processors Using Performance Counters" Proceedings of the 24th ACM International Conference on Supercomputing. 10.1145/1810085.1810108 ISBN: 978-1-4503-0018-6, 2010.

[BH 2007]

L. Barroso, U. Hölzle "The Case for Energy-Proportional Computing" IEEE Computer. 40. 10.1109/MC.2007.443 ISSN: 0018-9162, 2007.

[BHS 1998]

L. Benini, R. Hodgson, P. Siegel "System-level power estimation and optimization" Low Power Electronics and Design, 1998. Proceedings. 1998 International Symposium on. 10.1145/280756.280881 Print ISBN:1-58113-059-7, 1998.

[BJ 2003]

R. Bergamaschi, Y. Jiang "State-based power analysis for systems-on-chip" Design Automation Conference, 2003. Proceedings. 10.1109/DAC.2003.1219096 Print ISBN:1-58113-688-9, 2003.

[BJ 2007]

W. Bircher, L. John "Complete System Power Estimation: A Trickle-Down Approach Based on Performance Events" Performance Analysis of Systems Software, 2007. ISPASS 2007. IEEE International Symposium on. 10.1109/ISPASS.2007.363746 Print ISBN:1-4244-1082-7, 2007.

[BJ 2012]

W. Bircher, L. John "Complete System Power Estimation Using Processor Performance Events" Computers, IEEE Transactions on. 61, Heft 4. 10.1109/TC.2011.47 ISSN :0018-9340, 2012.

[BKW et al. 2003]

F. Bellosa, S. Kellner, M. Waitz, A. Weissel "Event-Driven Energy Accounting for Dynamic Thermal Management" In Proceedigns of COLP 2003, 2003.

[BLR et al. 2005]

N. Bansal, K. Lahiri, A. Raghunathan, S. Chakradhar "Power monitors: a framework for system-level power estimation using heterogeneous power models" VLSI Design, 2005. 18th International Conference on. 10.1109/ICVD.2005.138, 2005.

[BM 1995]

L. Benini, G. de Micheli "State assignment for low power dissipation" Solid-State Circuits, IEEE Journal of, Heft 3. 10.1109/4.364440 ISSN :0018-9200, 1995.

[BM 2001]

D. Brooks, M. Martonosi "Dynamic thermal management for high-performance microprocessors" High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on. 10.1109/HPCA.2001.903261 ISSN :1530-0897, 2001.

[BM 2012]

R. Basmadjian, H. de Meer "Evaluating and modeling power consumption of multi-core processors" Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy), 2012 Third International Conference on, 2012.

[Bor 1999]

S. Borkar "Design challenges of technology scaling" Micro, IEEE. 19, Heft 4. 10.1109/40.782564 ISSN :0272-1732, 1999.

[BR 2003]

B. Brock, K. Rajamani "Dynamic power management for embedded systems [SOC design]" SOC Conference, 2003. Proceedings. IEEE International [Systems-on-Chip]. 10.1109/SOC.2003.1241556 Print ISBN:0-7803-8182-3, 2003.

[BR 2012]

R. Balakrishnan, K. Ranganathan "A Textbook of Graph Theory". Springer New York, NY. 2nd ed. 2012, 2012.

[BS 1976]

F. Baskett, A. Smith "Interference in Multiprocessor Computer Systems with Interleaved Memory" New York, NY, USA, Heft 6. 10.1145/360238.360243 ISSN 0001-0782, 1976.

[BTM 2000]

D. Brooks, V. Tiwari, M. Martonosi "Wattch: a framework for architectural-level power analysis and optimizations" Computer Architecture, 2000. Proceedings of the 27th International Symposium on ISSN : 1063-6897, 2000.

[BWP et al. 2010]

T. Breen, E. Walsh, J. Punch, A. Shah, C. Bash "From chip to cooling tower data center modeling: Part I Influence of server inlet temperature and temperature rise across cabinet" Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), 2010 12th IEEE Intersociety Conference on. 12. 10.1109/ITHERM.2010.5501421 Print ISBN:978-1-4244-5342-9, 2010.

[CBB et al. 2010]

A. Castagnetti, C. Belleudy, S. Bilavarn, M. Auguin "Power Consumption Modeling for DVFS Exploitation" Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on. 10.1109/DSD.2010.55 Print ISBN: 978-1-4244-7839-2, 2010.

[CDS 2003]

J. Chen, M. Dubois, P. Stenstrom "Integrating complete-system and user-level performance/power simulators: the SimWattch approach" Performance Analysis of Systems and Software, 2003. ISPASS. 2003 IEEE International Symposium on. 10.1109/ISPASS.2003.1190227 Print ISBN:0-7803-7756-7, 2003.

[CDS 2007]
J. Chen, M. Dubois, P. Stenstrom "SimWattch: Integrating Complete-System and User-Level Performance and Power Simulators" Micro, IEEE, Heft 4. 10.1109/MM.2007.73 ISSN :0272-1732, 2007.

[Cha 2005]
N. Chang "In-House Tools for Low-Power Embedded Systems" Embedded Software and Systems. 10.1007/11535409_7 Print ISBN978-3-540-28128-3, 2005.

[Che 2006]
J. Chen "An Advanced Cache Power Model for An Embedded Processor using SLEEP Methodology" CAS-MS-2006-01, 2006.

[Che 2015]
B. Chen "Graph Theory" Hong Kong, 2015.

[CHW et al. 2010]
M. Cancian, Hauck, Jean Carlo Rossa, Wangenheim, Christiane Gresse von, R. Rabelo "Discovering Software Process and Product Quality Criteria in Software as a Service" Product-Focused Software Process Improvement. 6156. 10.1007/978-3-642-13792-1_19 Print ISBN 978-3-642-13791-4, 2010.

[CJ 2010]
Chakravarthy Akella, Jesus Yepez "Intel Power Management for Embedded Applications", 2010.

[CKS et al. 2007]
J. Choi, Y. Kim, A. Sivasubramaniam, J. Srebric, Q. Wang, J. Lee "Modeling and Managing Thermal Profiles of Rack-mounted Servers with ThermoStat" High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on. 13. 10.1109/HPCA.2007.346198 Print ISBN:1-4244-0805-9, 2007.

[CL 2008]
C. Cassandras, S. Lafortune "Introduction to discrete event systems". Springer New York, NY. 2. ed. 978-0-387-33332-8, 2008.

[Com 2013]
Comscore "Google Statistics", 12.07.2013, http://www.statisticbrain.com/google-searches/.

[Con 2012]
Constine Josh "How Big Is Facebook's Data? 2.5 Billion Pieces Of Content And 500+ Terabytes Ingested Every Day", 2012, http://techcrunch.com/2012/08/22/how-big-is-facebooks-data-2-5-billion-pieces-of-content-and-500-terabytes-ingested-every-day/.

[CPI et al. 2009]
K. Cameron, K. Pruhs, S. Irani, P. Ranganathan, D. Brooks "Report of the Science of Power Management Workshop" Arlington, 2009.

[CYR 2012]
Chia-Jung Chen, Yi-Sheng Liu, Rong-Guey Chang "DCSim: Design Analysis on Virtualization Data Center" Ubiquitous Intelligence Computing and 9th International Conference on Autonomic Trusted Computing (UIC/ATC), 2012 9th International Conference on. 10.1109/UIC-ATC.2012.66 Print ISBN:978-1-4673-3084-8, 2012.

[DAH et al. 2007]
Dam Sunwoo, H. Al-Sukhni, J. Holt, D. Chiou "Early Models for System-Level Power Estimation" Microprocessor Test and Verification, 2007. MTV '07. Eighth International Workshop on. 10.1109/MTV.2007.8 Print ISBN:978-0-7695-3241-7, 2007.

[DAJ et al. 2000]
Dejan Milojicic, Alan Messer, James Shau, Guangrui Fu, Alberto Munoz "Increasing Relevance of Memory Hardware Errors A Case for Recoverable Programming Models" Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system. 9. 10.1145/566726.566749, 2000.

[DBN et al. 2005]
David Wang, Brinda Ganesh, Nuengwong Tuaycharoen, Kathleen Baynes, Aamer Jaleel, Bruce Jacob "DRAMsim: A Memory System Simulator" ACM SIGARCH Computer Architecture News - Special issue: dasCMP'05. 33. 10.1145/1105734.1105748, 2005.

[Deb 2002]
K. Deb "Multi-objective optimization using evolutionary algorithms". Wiley Chichester [u.a.]. [Reprint]. 0-471-87339-X, 2002.

[DEP et al. 2009]
David Watts, Erwan Auffret, Phillip Dundas, Mark Kapoor "Tuning IBM System x Servers for Performance" International Business Machines Corporation. 6, 2009.

[DFM 2000]
J. Desharnais, M. Frappier, A. Mili "Handbook on electronic commerce". Springer Berlin. 978-3-540-25472-0, 2000.

[Don 2006]
J. Donato "Dual-Core Intel® Xeon® Processor 5100 Series" Reference Number: 313357-001, 2006.

[Don 2010]
E. Donald E. Porter "Understanding Transactional Memory Performance" Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on E-ISBN: 978-1-4244-6024-3, 2010.

[Dre 2006]
R. Dreier "Verteilte Ausführung hybrider System-Modelle auf heterogenen Rechnerplattformen", PhD thesis, 2006.

[DSH 2005]
M. Dellnitz, O. Schütze, T. Hestermeyer "Covering Pareto Sets by Multilevel Subdivision Techniques", Heft 1. 10.1007/s10957-004-6468-7, 2005.

[EG 2000]
G. Engels, Groenewegen Luuk "Object-oriented modeling: a roadmap". 10.1145/336512.336541 ICSE '00 Proceedings of the Conference on The Future of Software Engineering, 2000.

[EL 2009]

J. Ezekiel, A. Lomuscio "Combining Fault Injection and Model Checking to Verify Fault Tolerance in Multi-agent Systems" Richland, SC. Volume 1 ISBN: 978-0-9817381-6-1, 2009.

[EM 2010]

J. Encizo, D. Meyer "System x Server Optimization & Troubleshooting Best Practices Overview" IBM System x Technical University, 2010.

[Erd 2013]

H. Erden "Experimental and Analytical Investigation of the Transient Thermal Response of Air Cooled Data Centers", Syracuse University, NY, 2013.

[ERK 2006]

D. Economou, S. Rivoire, C. Kozyrakis "Full-system power analysis and modeling for server environments" In Workshop on Modeling Benchmarking and Simulation (MOBS), 2006.

[ES 2003]

A. Eiben, J. Smith "Introduction to evolutionary computing". Springer Berlin [u.a.]. 3-540-40184-9, 2003.

[Fac 2012]

I. Facebook "Amendment No. 4 to Form S-1 - Registration Statement", 2012, http://www.sec.gov/Archives/edgar/data/1326801/000119312512175673/d287954ds1a.htm.

[FBP et al. 2014]

A. Floratou, F. Bertsch, J. Patel, G. Laskaris "Towards Building Wind Tunnels for Data Center Design" Proc. VLDB Endow. 7, Heft 9. 10.14778/2732939.2732950, 2014.

[FCM 2014]

B. Fischer, C. Cech, H. Muhr "Power Modeling and Analysis in Early Design Phases" Proceedings of the Conference on Design, Automation \& Test in Europe ISBN: 978-3-9815370-2-4, 2014.

[FM 2002]

K. Flautner, T. Mudge "Vertigo: Automatic Performance-Setting for Linux" Proceedings of the 5th Symposium on Operating Systems Design and implementation. 10.1145/1060289.1060300 ISBN: 978-1-4503-0111-4, 2002.

[FOG 2008]

S. Ferson, W. Oberkampf, L. Ginzburg "Model validation and predictive capability for the thermal challenge problem", 29-32. 10.1016/j.cma.2007.07.030, 2008.

[FRK et al. 2005]

W. Felter, K. Rajamani, T. Keller, C. Rusu "A Performance-conserving Approach for Reducing Peak Power Consumption in Server Systems" Proceedings of the 19th Annual International Conference on Supercomputing. 10.1145/1088149.1088188 ISBN:1-59593-167-8, 2005.

[Fuj 2009], [Fuj 2011], [Fuj 2012], [Fuj 2013], [Fuj 2014], [Fuj 2015]

Fujitsu Technology Solutions GmbH, Developer, "Internal statement”, Internal Documents and Presentations, 2009-2015.

[Fuj 2010]

Fujitsu "CO2-Footprint of PRIMERGY RX200 / TX300 S5 Servers", 2010.

[FWB 2007]

X. Fan, W.-D. Weber, L. Barroso "Power Provisioning for a Warehouse-sized Computer" Proceedings of the 34th Annual International Symposium on Computer Architecture. 35, Heft 2. 10.1145/1273440.1250665 ISBN: 978-1-59593-706-3, 2007.

[FZL et al. 2012]

K. Fang, H. Zheng, J. Lin, Z. Zhang, Z. Zhu "Mini-Rank: A Power-Efficient DDRx DRAM Memory Architecture" Computers, IEEE Transactions on. 10.1109/TC.2012.240, 2012.

[GBC et al. 2001]

Gunther, Binns, Carmean, Hall "Managing the Impact of Increasing Microprocessor Power Consumption" Intel Technology Journal. 5, 2001.

[GC 2000]
M. Gen, R. Cheng "Genetic algorithms and engineering optimization". Wiley New York [u.a.]. 0-471-31531-1, 2000.

[Gee 2004]
H. Geering "Regelungstechnik". Springer Berlin. 6. 3-540-40507-0, 2004.

[GFN et al. 2006]
Gergö Lovasz, Florian Niedermeier, Nasir Ali, Robert Basmadjian, Hermann de Meer "Modeling Power Consumption of the G-Lab Platform to Enable an Energy-Efficient Provision of Services" Leibnitz Universität Hannover, Fakultät für Elektrotechnik und Informatik, Institut für Kommunikationstechnik Project G-Lab Ener-G, 2006.

[GHD et al. 2009]
A. Gandhi, M. Harchol-Balter, R. Das, C. Lefurgy "Optimal Power Allocation in Server Farms" Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems. 37, Heft 1. 10.1145/2492101.1555368, 2009.

[GKG 2012]
M. Gottscho, A. Kagalwalla, P. Gupta "Power Variability in Contemporary DRAMs". 4, Heft 2. 10.1109/LES.2012.2192414 ISSN 1943-0663, 2012.

[Gre 1994]
P. Greenawalt "Modeling power management for hard disks" Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 1994., MASCOTS '94., Proceedings of the Second International Workshop on. 2. 10.1109/MASCOT.1994.284446 Print ISBN:0-8186-5292-6, 1994.

[GSI et al. 2002]
S. Gurumurthi, A. Sivasubramaniam, M. Irwin, N. Vijaykrishnan, M. Kandemir "Using complete machine simulation for software power estimation: the SoftWatt approach" High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium on. 10.1109/HPCA.2002.995705, 2002.

[GSK et al. 2009]

R. Garg, Seung Woo Son, M. Kandemir, P. Raghavan, R. Prabhakar "Markov Model Based Disk Power Management for Data Intensive Workloads" Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on. 10.1109/CCGRID.2009.67 E-ISBN :978-0-7695-3622-4, 2009.

[GTW 2006]

Greenberg, Tschudi, Weale "Self Benchmarking Guide for Data Center Energy Performance" Berkeley National Laboratory. 1, 2006.

[GX 2010]

J. Gong, C.-Z. Xu "A Gray-Box Feedback Control Approach for System-Level Peak Power Management" Parallel Processing (ICPP), 2010 39th International Conference on. 10.1109/ICPP.2010.63 Print ISBN: 978-1-4244-7913-9, 2010.

[Hag 2009]

O. Hagendorf "Simulation Based Parameter and Structure Optimisation of Discrete Event Systems", PhD thesis, 2009.

[Ham 1994]

D. Hamby "A review of techniques for parameter sensitivity analysis of environmental models". 32, Heft 2. 10.1007/BF00547132 Online ISSN1573-2959, 1994.

[Han 2007]

H. Hanson "Coordinated Power, Energy, and Temperature Management" The University of Texas at Austin, Electrical & Computer Engineering ISBN: 978-0-549-16370-1, 2007.

[HCE et al. 2011]

Howard David, Chris Fallin, Eugene Gorbatov, Ulf R. Hanebutte, Onur Mutlu "Memory power management via dynamic voltage/frequency scaling" ICAC '11 Proceedings of the 8th ACM international conference on Autonomic computing. 10.1145/1998582.1998590, 2011.

[HCG et al. 2006]

T. Heath, A. Centeno, P. George, L. Ramos, Y. Jaluria, R. Bianchini "Mercury and Freon: Temperature Emulation and Management for Server Systems" Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, Heft 11. 10.1145/1168918.1168872 ISBN:1-59593-451-0, 2006.

[HCY et al. 2005]

J. Huang, T. Chen, M. Ye, Y. Lian "The Modeling for Dynamic Power Management of Embedded Systems" Embedded Software and Systems. 3605. 10.1007/11535409_67 Print ISBN978-3-540-28128-3, 2005.

[Her 1998]

S. Herrod "Using Complete Machine Simulation to Understand Computer System Behavior, PhD thesis" Stanford University California, 1998.

[Hex 2003]

R. Hexel "FITS: A Fault Injection Architecture for Time-triggered Systems" Darlinghurst, Australia, Australia. 16 ISBN: 0-909-92594-1, 2003.

[HFS 2010]

R. Hintemann, K. Fichter, L. Stobbe "Materialbestand der Rechenzentren in Deutschland -" Dessau-Roßlau ISSN 1862-4804, 2010.

[HHM 2008]

J. Harris, J. Hirst, M. Mossinghoff "Combinatorics and Graph Theory". Springer New York. 978-0-387-79710-6, 2008.

[HIM et al. 2013]

Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., Toshiba Corporation "Advanced Configuration and Power Interface Specification". Revision 5.0 Errata A, 2013.

[HJZ et al. 2008]

Hongzhong Zheng, Jiang Lin, Zhao Zhang, Eugene Gorbatov, Howard David, Zhichun Zhu "Mini-Rank: Adaptive DRAM Architecture for Improving Memory Power Efficiency" Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on. 41. 10.1109/MICRO.2008.4771792 Date of Conference: 8-12 Nov. 2008, 2008.

[HK 2003]

C.-H. Hsu, U. Kremer "The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction" Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation. 38, Heft 5. 10.1145/780822.781137 ISBN:1-58113-662-5, 2003.

[HKG et al. 2007]

H. Hanson, S. Keckler, S. Ghiasi, K. Rajamani, F. Rawson, J. Rubio "Thermal response to DVFS: analysis with an Intel Pentium M" Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on. 10.1145/1283780.1283827 E-ISBN :978-1-59593-709-4, 2007.

[HLH et al. 2012]

M. Hagan, J. Lusky, T. Hoang, S. Walsh "The Top 9 Mistakes in Data Center Planning" Schneider Electric – Data Center Science Center, 2012.

[HMU 2001]

J. Hopcroft, R. Motwani, J. Ullman "Introduction to automata theory, languages, and computation". Addison-Wesley Boston [u.a.]. 2. ed. 0-201-44124-1, 2001.

[HRR et al. 2007]

H. Hanson, K. Rajamani, J. Rubio, S. Ghiasi, Rawson, Freeman "Benchmarking for Power and Performance" IBM Austin Research Laboratory, 2007.

[HS 1997]

A. Howe, G. Somlo "Modelling discrete event sequences as state transition diagrams". Springer-Verlag Berlin Heidelberg. 978-3-540-63346-4, 1997.

[HS 2007]
Heather Hanson, Stephen W. Keckler "Power and Thermal Characteristics of a Pentium M System" The University of Texas at Austin, Electrical & Computer Engineering, 2007.

[HSR et al. 2008]
A. Hylick, R. Sohan, A. Rice, B. Jones "An Analysis of Hard Drive Energy Consumption" 2008 IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems. 10.1109/MASCOT.2008.4770567 ISSN 1526-7539, 2008.

[HSW et al. 2004]
N. Hardavellas, S. Somogyi, T. Wenisch, R. Wunderlich, S. Chen, J. Kim, B. Falsafi, J. Hoe, A. Nowatzyk "SimFlex: A Fast, Accurate, Flexible Full-system Simulation Framework for Performance Evaluation of Server Architecture" ACM SIGMETRICS Performance Evaluation Review - Special issue on tools for computer architecture research. 31, Heft 4. 10.1145/1054907.1054914, 2004.

[HXL et al. 2002]
Hang-Sheng Wang, Xinping Zhu, Li-Shiuan Peh, S. Malik "Orion: a power-performance simulator for interconnection networks" Microarchitecture, 2002. (MICRO-35). Proceedings. 35th Annual IEEE/ACM International Symposium on. 10.1109/MICRO.2002.1176258 Print ISBN: 0-7695-1859-1, 2002.

[IAE et al. 2011]
T. Inoue, A. Aikebaier, T. Enokido, M. Takizawa "A Power Consumption Model of a Storage Server" Network-Based Information Systems (NBiS), 2011 14th International Conference on. 10.1109/NBiS.2011.64, 2011.

[IBS et al. 2012]
M. Ibrahim, S. Bhopte, B. Sammakia, B. Murray, M. Iyengar, R. Schmidt "Effect of Transient Boundary Conditions and Detailed Thermal Modeling of Data Center Rooms" Components, Packaging and Manufacturing Technology, IEEE Transactions on, Heft 2. 10.1109/TCPMT.2011.2175926 ISSN :2156-3950, 2012.

[IGB et al. 2010]
M. Ibrahim, S. Gondipalli, S. Bhopte, B. Sammakia, B. Murray, K. Ghose, M. Iyengar, R. Schmidt "Numerical modeling approach to dynamic data center cooling" Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), 2010 12th IEEE Intersociety Conference on. 10.1109/ITHERM.2010.5501335 Print ISBN:978-1-4244-5342-9, 2010.

[IIE et al. 2011]
T. Inoue, M. Ikeda, T. Enokido, A. Aikebaier, M. Takizawa "A Power Consumption Model for Storage-based Applications" Complex, Intelligent and Software Intensive Systems (CISIS), 2011 International Conference on. 10.1109/CISIS.2011.101 E-ISBN :978-0-7695-4373-4, 2011.

[IIZ et al. 2007]
Iyer, Illikkal, Zhao, Makineni, Newell, Moses, Apparao "Datacenter-on-Chip Architectures: Tera-scale Opportunities and Challenges in Intel's Manufacturing Environment" Intel Technology Journal. 11, Heft 03. 10.1535/itj.1103.06 ISSN1535-864X, 2007.

[IM 2002]
IBM, MontaVista Software "Dynamic Power Management for Embedded Systems" IBM and MontaVista Software, 2002.

[IM 2003]
C. Isci, M. Martonosi "Runtime power monitoring in high-end processors: methodology and empirical data" Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on. 10.1109/MICRO.2003.1253186, 2003.

[Int 2006]
Intel Corporation "Intel Architecture and Silicon Cadence", 2006.

[Int 2013]
Intel Corporation "Intel® Public Roadmap Desktop, Mobile & Data Center", 2013.

[Int 2014]
Intel Corporation "Intel® Xeon® Processor E5-1600/E5-2600/E5-4600 v2 Product Families", 2014.

[Int 2015]
Intersil Corporation "TB379: Thermal Characterization of Packaged Semiconductor Devices" Intersil Corporation, 2015.

[ISO 2011]
ISO/IEC "ISO/IEC 25010:2011(en) - Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models", 2011.

[ISS et al. 2012]
M. Ibrahim, S. Shrivastava, B. Sammakia, K. Ghose "Thermal mass characterization of a server at different fan speeds" Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), 2012 13th IEEE Intersociety Conference on. 13. 10.1109/ITHERM.2012.6231467 Print ISBN:978-1-4244-9533-7, 2012.

[Jau 2011]
E. Jaureguialzo "PUE: The Green Grid metric for evaluating the energy efficiency in DC (Data Center). Measurement method using the Power Demand" Telecommunications Energy Conference (INTELEC), 2011 IEEE 33rd International. 10.1109/INTLEC.2011.6099718 ISBN: 978-1-4577-1249-4, 2011.

[JCX 2008]
Jinsong Ji, Chao Wang, Xuehai Zhou "System-Level Early Power Estimation for Memory Subsystem in Embedded Systems" Embedded Computing, 2008. SEC '08. Fifth IEEE International Symposium on. 10.1109/SEC.2008.48 Print ISBN:978-0-7695-3348-3, 2008.

[JD 2012]
John Gantz, David Reinsel "THE DIGITAL UNIVERSE IN 2020", 2012.

[JGM 2003]
N. Julien, S. Gailhard, E. Martin "Low Power Synthesis Methodology with Data Format Optimization Applied on a DWT" Journal of VLSI signal processing systems for signal, image and video technology, Heft 2. 10.1023/A:1023660801499 Print ISSN: 0922-5773, 2003.

[JLS et al. 2003]

N. Julien, J. Laurent, E. Senn, E. Martin "Power consumption modeling and characterization of the TI C6201", Heft 5. 10.1109/MM.2003.1240211 ISSN: 0272-1732, 2003.

[JM 2001]

R. Joseph, M. Martonosi "Run-time Power Estimation in High Performance Microprocessors" Proceedings of the 2001 International 2001. 10.1145/383082.383119 ISBN:1-58113-371-5, 2001.

[JPL et al. 2014]

E. Jo, D. Pan, J. Liu, L. Butler "A Simulation and Emulation Study of SDN-based Multipath Routing for Fat-tree Data Center Networks" Proceedings of the 2014 Winter Simulation Conference, 2014.

[Jun 1999]

U. Jung "Ventilatoren-Fibel". Promotor Verlag Karlsruhe. 3-00-003293-2, 1999.

[JVG 2010]

M. Jonas, G. Varsamopoulos, S. Gupta "Non-invasive Thermal Modeling Techniques Using Ambient Sensors for Greening Data Centers" Parallel Processing Workshops (ICPPW), 2010 39th International Conference on. 10.1109/ICPPW.2010.67 Print ISBN:978-1-4244-7918-4, 2010.

[KAM et al. 2002]

N. Kim, T. Austin, T. Mudge, D. Grunwald "Challenges for Architectural Level Power Modeling" Power Aware Computing. 10.1007/978-1-4757-6217-4_16 Print ISBN978-1-4419-3382-9, 2002.

[KBK 2012]

Karthik Chandrasekar, Benny Akesson, Kees Goossens "Run-Time Power-Down Strategies for Real-Time SDRAM Memory Controllers" DAC '12 Proceedings of the 49th Annual Design Automation Conference. 10.1145/2228360.2228538 ISSN :0738-100X, 2012.

[KBS et al. 2010]

J. Koomey, S. Berard, M. Sanchez, H. Wong "Implications of Historical Trends in the Electrical Efficiency of Computing" Annals of the History of Computing, IEEE. 33. 10.1109/MAHC.2010.28 Annals of the History of Computing, IEEE, 2010.

[KCB et al. 2013]

C. Karthik, Christian Weis, Benny Akesson, Norbert Wehn, Kees Goossens "Towards Variation-Aware System-Level Power Estimation of DRAMs: An Empirical Approach" DAC'13, 2013.

[KFK 2008]

J. Kaplan, W. Forrest, N. Kindler "Revolutionizing Data Center Energy Efficiency" McKinsey & Company, 2008.

[KGK et al. 2005]

R. Kotla, S. Ghiasi, T. Keller, F. Rawson "Scheduling processor voltage and frequency in server and cluster systems" Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International. 10.1109/IPDPS.2005.392 Print ISBN:0-7695-2312-9, 2005.

[KGS 2008]

S. Kounev, I. Gorton, K. Sachs "Performance Evaluation: Metrics, Models and Benchmarks". Springer Berlin Heidelberg SPEC International Performance Evaluation Workshop, SIPEW 2008. 5119. 9783540698142, 2008.

[KIM et al. 2011]

Karthik Elangovan, Ivan Rodero, Manish Parashar, Francesc Guim, Isaac Hernandez "Adaptive Memory Power Management Techniques for HPC Workloads" High Performance Computing (HiPC), 2011 18th International Conference on, 2011.

[KJC et al. 2014]

M. Kim, Y. Ju, J. Chae, M. Park "A Simple Model for Estimating Power Consumption of a Multicore Server System" International Journal of Multimedia and Ubiquitous Engineeerign. 9, Heft 2. 10.14257/ijmue.2014.9.2.15, 2014.

[KKK et al. 2012]
M. Kim, P. Kumar, H. Kim, B. Brett "Predicting Potential Speedup of Serial Code via Lightweight Profiling and Emulations with Memory Performance Model" IPDPS '12 Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium. 26. 10.1109/IPDPS.2012.128, 2012.

[KLL et al. 2008]
A. Kumar, Li Shang, Li-Shiuan Peh, N. Jha "System-Level Dynamic Thermal Management for High-Performance Microprocessors" Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, Heft 1. 10.1109/TCAD.2007.907062 ISSN :0278-0070, 2008.

[Koo 2011]
J. Koomey, "GROWTH IN DATA CENTER ELECTRICITY USE 2005 TO 2010", Heft 3, 2011.

[KRS et al. 2014]
D. King, M. Ross, M. Seymour, T. Gregory "Comparative analysis of data center design showing the benefits of server level simulation models" Semiconductor Thermal Measurement and Management Symposium (SEMI-THERM), 2014 30th Annual. 10.1109/SEMI-THERM.2014.6892238, 2014.

[KS 2005]
Kyeong-Jae Lee, K. Skadron "Using performance counters for runtime temperature sensing in high-performance processors" Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International. 19. 10.1109/IPDPS.2005.448 Print ISBN:0-7695-2312-9, 2005.

[KSH et al. 2009]
D. Khalil, D. Sinha, Hai Zhou, Y. Ismail "A Timing-Dependent Power Estimation Framework Considering Coupling" Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, Heft 6. 10.1109/TVLSI.2008.2008739 ISSN :1063-8210, 2009.

[KTL et al. 2013]
G. Keller, M. Tighe, H. Lutfiyya, M. Bauer "DCSim: A data centre simulation tool" Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on Print ISBN:978-1-4673-5229-1, 2013.

[Kur 2008]
K. Kurbel "The Making of Information Systems". Springer-Verlag Berlin Heidelberg Berlin, Heidelberg. 978-3-540-79260-4, 2008.

[KZM 2001]
S. Kaxiras, Zhigang Hu, M. Martonosi "Cache decay: exploiting generational behavior to reduce cache leakage power" Computer Architecture, 2001. Proceedings. 28th Annual International Symposium on. 28. 10.1109/ISCA.2001.937453 ISSN :1063-6897, 2001.

[Lan 1996]
P. Landman "High-level Power Estimation" Proceeding ISLPED '96 Proceedings of the 1996 international symposium on Low power electronics and design ISBN:0-7803-3571-6, 1996.

[Lan 2007]
R. Lantz "Parallel SimOS: Performance and Scalability for Large System Simulation, PhD thesis" Stanford University California, 2007.

[LB 2005]
Loucks, Beek "WATER RESOURCES SYSTEMS PLANNING AND MANAGEMENT". United Nations Educational UNESCO PUBLISHING. 92-3-103998-9, 2005.

[LCW 2009]
P. Lu, Y. Che, Z. Wang "A Framework for Effective Memory Optimization of High Performance Computing Applications" High Performance Computing and Communications, 2009. HPCC '09. 11th IEEE International Conference on. 11. 10.1109/HPCC.2009.60 Date of Conference: 25-27 June 2009, 2009.

[LD 2000]
V. Le, S. Donald "Understanding Data Flow Diagrams" Proceedings of the 47th annual, 2000.

[LEU et al. 2010]
Le Howard David, Eugene Gorbatov, Ulf R. Hanebutte, Rahul Khanaa, Christian Kruse "RAPL: memory power estimation and capping" Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on. 10.1145/1840845.1840883, 2010.

[Lin 2001]

J. Lind "Iterative software engineering for multiagent systems". Springer Berlin. ISBN 3-540-42166-1, 2001.

[Lin 2009]

Linaege Power "Thermal Characterization Process For Open-Frame Board Mounted Power Modules", 2009.

[Liu 2011]

H. Liu "A Measurement Study of Server Utilization in Public Clouds" Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on. 10.1109/DASC.2011.87 Print ISBN:978-1-4673-0006-3, 2011.

[LJ 2003]

T. Li, L. John "Run-time Modeling and Estimation of Operating System Power Consumption" SIGMETRICS '03 Proceedings of the 2003 ACM SIGMETRICS, Heft 1. 10.1145/885651.781048 ISBN:1-58113-664-1, 2003.

[LJS et al. 2004]

J. Laurent, N. Julien, E. Senn, E. Martin "Functional level power analysis: an efficient approach for modeling the power consumption of complex processors". 1. 10.1109/DATE.2004.1268921 ISSN: 1530-1591, 2004.

[LK 2000]

A. Law, W. Kelton "Simulation modeling and analysis" Boston [u.a.]. 3. ed, 2000.

[Lon 2012]

B. London "A model-based system engineering framework for concept development" Massachusetts Institute of Technology, 2012.

[LPT 1994]

P. Larsen, N. Plat, H. Toetenel "A Formal Semantics of Data Flow Diagrams". Volume 6, Issue 6. 10.1007/BF03259387 ISSN: 0934-5043, 1433-299X, 1994.

[LR 1996]

D. Lidsky, J. Rabaey "Early power exploration-a World Wide Web application" Design Automation Conference Proceedings 1996, 33rd. 33. 10.1109/DAC.1996.545539 ISSN :0738-100X, 1996.

[LRR et al. 2004]

Lin Zhong, S. Ravi, A. Raghunathan, N. Jha "Power estimation for cycle-accurate functional descriptions of hardware" Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on. 10.1109/ICCAD.2004.1382659 Print ISBN:0-7803-8702-3, 2004.

[LS 1994]

D. Liu, C. Svensson "Power consumption estimation in CMOS VLSI chips" Solid-State Circuits, IEEE Journal of, Heft 6. 10.1109/4.293111 ISSN :0018-9200, 1994.

[LSQ et al. 2014]

X. Liu, L. Shen, C. Qian, Z. Wang "Dynamic Power Estimation with Hardware Performance Counters Support on Multi-core Platform" Advanced Computer Architecture. 10.1007/978-3-662-44491-7_14 Print ISBN978-3-662-44490-0, 2014.

[LU 2009]

Luiz André Barroso, Urs Hölzle "An Introduction to the Design of Warehouse-Scale Machines" SYNTHESIS LECTURES ON COMPUTER ARCHITECTURE. 6. 10.2200/S00193ED1V01Y200905CAC006 ISBN: 9781598295566, 2009.

[LYB et al. 2012]

Lei Jiang, Youtao Zhang, Bruce R Childers, Jun Yang "FPB: Fine-grained Power Budgeting to Improve Write Throughput of Multi-level Cell Phase Change Memory" MICRO 12 Proceesdings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture. 45. 10.1109/MICRO.2012.10, 2012.

[LZZ et al. 2007]

J. Lin, H. Zheng, Z. Zhu, H. David, Z. Zhang "Thermal Modeling and Management of DRAM Memory Systems" ISCA '07 Proceedings of the 34th annual international symposium on Computer architecture. 34, Heft 2. 10.1145/1273440.1250701 ISBN: 978-1-59593-706-3, 2007.

[MAC et al. 2011]
J. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. Snoeren, R. Gupta "Evaluating the Effectiveness of Model-based Power Characterization" Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference, 2011.

[Mat 2009]
Matthew E. Tolentino "Managing Memory for Power Performance and Thermal Efficiency" Blacksburg, VA URN: etd-02242009-162329, 2009.

[Mat 2011]
P. Mathew "Recommendations for Measuring and Reporting Overall Data Center Efficiency", 2011.

[MB 2006]
A. Merkel, F. Bellosa "Balancing Power Consumption in Multiprocessor Systems" Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006, Heft 4. 10.1145/1218063.1217974 ISBN:1-59593-322-0, 2006.

[MCE et al. 2002]
P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, B. Werner "Simics: A full system simulation platform" Computer. 35, Heft 2. 10.1109/2.982916, 2002.

[MHS et al. 2009]
D. Molka, D. Hackenberg, R. Schone, M. Muller "Memory Performance and Cache Coherency Effects on an Intel Nehalem Multiprocessor System" Parallel Architectures and Compilation Techniques, 2009. PACT '09. 18th International Conference on. 10.1109/PACT.2009.22, 2009.

[MHW 2002]
C. Mauer, M. Hill, D. Wood "Full-system Timing-first Simulation" Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, Heft 1. 10.1145/511399.511349 ISBN:1-58113-531-9, 2002.

[Mic 2007]
I. Micron Technology "TN-41-01: Calculating Memory System Power for DDR3" Micron Technology, Inc., 2007.

[MJW 2012]
D. Meisner, Junjie Wu, T. Wenisch "BigHouse: A simulation infrastructure for data center systems" Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium on. 10.1109/ISPASS.2012.6189204 Print ISBN:978-1-4673-1143-4, 2012.

[MKO et al. 2002]
H. Mizuno, H. Kobayashi, T. Onoye, I. Shirakawa "Power estimation at architecture level for embedded systems" Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on. 10.1109/ISCAS.2002.1011028, 2002.

[MMA 2014]
N. Moustafa, M. Mashaly, M. Ashour "Modeling and simulation of energy-efficient cloud data centers" Engineering and Technology (ICET), 2014 International Conference on. 10.1109/ICEngTechnol.2014.7016745, 2014.

[MNR 2007]
F. Mesa-Martinez, J. Nayfach-Battilana, J. Renau "Power Model Validation Through Thermal Measurements" Proceedings of the 34th Annual International Symposium on Computer Architecture, Heft 2. 10.1145/1273440.1250700 ISBN: 978-1-59593-706-3, 2007.

[Moo 1965]
G. Moore "Cramming more components onto integrated circuits" Solid-State Circuits Society Newsletter, IEEE. volume 38. 10.1109/N-SSC.2006.4785860 Solid-State Circuits Society Newsletter, IEEE, 1965.

[Mos 2005]
S. Mostaghim "Multi-objective evolutionary algorithms – PhD Thesis", 2005.

[MPL 2009]
D. Molaro, H. Payer, D. Le Moal "Tempo: Disk drive power consumption characterization and modeling" Consumer Electronics, 2009. ISCE '09. IEEE 13th International Symposium on. 13. 10.1109/ISCE.2009.5156863 E-ISBN :978-1-4244-2976-9, 2009.

[MSB et al. 2005]
Martin, Milo M. K., D. Sorin, B. Beckmann, M. Marty, M. Xu, A. Alameldeen, K. Moore, M. Hill, D. Wood "Multifacet’s General Execution-driven Multiprocessor Simulator (GEMS) Toolset" SIGARCH Comput. Archit. News, Heft 4. 10.1145/1105734.1105747, 2005.

[MSD 2006]
Muhammad Sibghatullah, Syed Jafar Hussain, Dr. Syed Jamal Hussain "An Approach to Effective Product Development Life Cycle". 10.1109/ICET.2006.335975 Emerging Technologies, 2006. ICET '06. International Conference on, 2006.

[MXX 2011]
Ming Chen, Xiaorui Wang, Xue Li "Coordinating Processor and Main Memory for Efficient Server Power Control" ICS '11 Proceedings of the international conference on Supercomputing. 10.1145/1995896.1995917, 2011.

[Naj 1995]
F. Najm "Towards a High-level Power Estimation Capability" Proceedings of the 1995 International Symposium on Low Power Design. 10.1145/224081.224097 ISBN:0-89791-744-8, 1995.

[Neb 2014]
G. Nebuloni "Server Revenues Fall Further in EMEA in 4Q13 But Units Remain Stable" FRANKFURT and PRAGUE, 18.03.2014.

[New 2008]
L. Newcombe "Data centre energy efficiency metrics" Data Centre Specialist Group (DCSG), 2008.

[NKB et al. 2004]

Nam Sung Kim, T. Kgil, V. Bertacco, T. Austin, T. Mudge "Microarchitectural Power Modeling Techniques for Deep Sub-Micron Microprocessors" Low Power Electronics and Design, 2004. ISLPED '04. Proceedings of the 2004 International Symposium on Print ISBN:1-58113-929-2, 2004.

[NKN et al. 2002]

S. Nikolaidis, N. Kavvadias, P. Neofotistos, K. Kosmatopoulos, T. Laopoulos, L. Bisdounis "Instrumentation Set-up for Instruction Level Power Modeling". 10.1007/3-540-45716-X_8 Print ISBN: 978-3-540-44143-4, 2002.

[NXP 2010]

NXP Semiconductors "DDR memory module temp sensor with integrated SPD", 2010.

[Oli 2007]

A. Olivé "Conceptual Modeling of Information Systems". Springer Berlin Heidelberg. 978-3-540-39389-4, 2007.

[Ols 2000]

Olshausen "Aliasing", 2000.

[Osi 2010]

J. Osis "Model-Driven Domain Analysis and Software Development". Information Science Reference, IGI Global. 9781616928766, 2010.

[Paw 1990]

K. Pawlikowski "Steady-State Simulation of Queueing Processes - A Survey of Problems and Solutions" ACM Computing Surveys. 22. 10.1145/78919.78921 ISSN: 0360-0300, 1990.

[PBB et al. 2010]

J. Pflueger, K. Baker, M. Blackburn, T. Brey, B. Carter, M. Criss, T. Harvey, A. Hawkins, Z. Ortiz "A roadmap for the adoption of power-related features in servers" The Green Grid. 33, 2010.

[Pet 2012]

Pettey Christy "Gartner Identifies the Top 10 Strategic Technologies for 2012", 2012, http://www.gartner.com/newsroom/id/1826214.

[PS 2004]

C. Pereira, P. Sousa "A Method to Define an Enterprise Architecture using the Zachman Framework" ACM Symposium on Applied Computing. 10.1145/967900.968175 ISBN:1-58113-812-1, 2004.

[PV 2014]

Z. Pardey, J. VanGilder "Further exploration of a compact transient server model" Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), 2014 IEEE Intersociety Conference on. 10.1109/ITHERM.2014.6892433 ISSN :1087-9870, 2014.

[PZH et al. 2005]

Pu Liu, Zhenyu Qi, Hang Li, Lingling Jin, Wei Wu, S. Tan, Jun Yang "Fast thermal simulation for architecture level dynamic thermal management" Computer-Aided Design, 2005. ICCAD-2005. IEEE/ACM International Conference on. 10.1109/ICCAD.2005.1560145 Print ISBN:0-7803-9254-X, 2005.

[Qia 2011]

Qiang Zou "An analytical performance and power model based on the transition probability for hard disks" Awareness Science and Technology (iCAST), 2011 3rd International Conference on. 10.1109/ICAwST.2011.6163123 Print ISBN:978-1-4577-0887-9, 2011.

[Qih 2008]

N. Qihong "Experimentally Validated Multiscale Thermal Modeling of Electric Cabinets, PhD thesis" Georgia Institute of Technology, GA, 2008.

[QXY 2008]

Qi Zhu, Xiang Li, Yinan Wu "Thermal managerment of high power memory module for server platforms" Thermal and Thermomechanical Phenomena in Electronic Systems, 2008. ITHERM 2008. 11th Intersociety Conference on. 11. 10.1109/ITHERM.2008.4544319 Print ISBN:978-1-4244-1700-1, 2008.

[Rab 2010]
S. Rabinovich "Evaluating Measurement Accuracy". Springer New York, NY : Springer Science+Business Media, LLC New York, NY. 978-1-4419-1455-2, 2010.

[RAK et al. 2013]
R. Rodrigues, A. Annamalai, I. Koren, S. Kundu "A Study on the Use of Performance Counters to Estimate Power in Microprocessors" Circuits and Systems II: Express Briefs, IEEE Transactions on. 60, Heft 12. 10.1109/TCSII.2013.2285966 ISSN :1549-7747, 2013.

[RAM et al. 2009]
R. Romadhon, M. Ali, A. Mahdzir, Y. Abakr "Optimization of cooling systems in data centre by Computational Fluid Dynamics model and simulation" Innovative Technologies in Intelligent Systems and Industrial Applications, 2009. CITISIA 2009. 10.1109/CITISIA.2009.5224189 Print ISBN:978-1-4244-2886-1, 2009.

[Ran 2010]
P. Ranganathan "Recipe for Efficiency: Principles of Power-aware Computing" Commun. ACM. 53, Heft 4. 10.1145/1721654.1721673, 2010.

[Raw 2004]
F. Rawson "MEMPOWER: A Simple Memory Power Analysis Tool Set, Technical Report" IBM Austin Research Laboratory, 2004.

[RF 2009]
K. Rogoz, K. Figura "A Test Based Multidimensional Performance Model for a Mission Critical System Server" Software Testing Verification and Validation, 2009. ICST '09. International Conference on. 10.1109/ICST.2009.40 E-ISBN :978-0-7695-3601-9, 2009.

[RHA et al. 2007]
E. Rotem, J. Hermerding, C. Aviad, C. Harel "Temperature measurement in the Intel Core Duo Processor" Dans Proceedings of 12th International Workshop on Thermal investigations of ICs - THERMINIC 2006, Nice : France (2006), 2007.

[RHH et al. 2005]
Rong Luo, Hong Luo, Huazhong Yang, Yuan Xie "An instruction-level analytical power model for designing the low power systems on a chip". 2. 10.1109/ICASIC.2005.1611515 Print ISBN: 0-7803-9210-8, 2005.

[RHR et al. 2006]
K. Rajamani, H. Hanson, J. Rubio, S. Ghiasi, F. Rawson "Application-Aware Power Management" Workload Characterization, 2006 IEEE International Symposium on. 10.1109/IISWC.2006.302728 E-ISBN :1-4244-0509-2, 2006.

[RHW et al. 1995]
M. Rosenblum, S. Herrod, E. Witchel, A. Gupta "Complete Computer System Simulation: The SimOS Approach" IEEE Parallel Distrib. Technol., Heft 4. 10.1109/88.473612, 1995.

[Riv 2008]
S. Rivoire "Models and Metrics for Energy-Efficient Computer Systems, PhD thesis" Stanford University California, 2008.

[RL 2007]
P. Ranganathan, P. Leech "Simulating Complex Enterprise Workloads using Utilization Traces" In Proceedings of the Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW), 2007.

[RLG et al. 2008]
Rajamani, Lefurgy, Ghiasi, Rubio, Hanson, Keller "Power Management for Computer Systems and Datacenters", 2008.

[RLI et al. 2006]
P. Ranganathan, P. Leech, D. Irwin, J. Chase "Ensemble-level Power Management for Dense Blade Servers" Proceedings of the 33rd Annual International Symposium on Computer Architecture. 33, Heft 2. 10.1145/1150019.1136492 ISBN:0-7695-2608-X, 2006.

[RM 2005]
F. Ridruejo Perez, J. Miguel-Alonso "INSEE: An Interconnection Network Simulation and Evaluation Environment" Euro-Par 2005 Parallel Processing. 3648. 10.1007/11549468_111 Print ISBN978-3-540-28700-1, 2005.

[RMG et al. 2009]

E. Rotem, A. Mendelson, R. Ginosar, U. Weiser "Multiple clock and Voltage Domains for chip multi processors" Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on ISSN :1072-4451, 2009.

[RMN 2009]

F. Ridruejo, J. Miguel-Alonso, J. Navaridas "Full-system Simulation of Distributed Memory Multicomputers" Cluster Computing. 12, Heft 3. 10.1007/s10586-009-0086-y, 2009.

[RN 2011]

N. Rasmussen, S. Niles "Data Center Projects: Growth Model" Schneider Electric – Data Center Science Center, 2011.

[RRK 2008]

S. Rivoire, P. Ranganathan, C. Kozyrakis "A Comparison of High-level Full-system Power Models" Proceedings of the 2008 Conference on Power Aware Computing and Systems, 2008.

[RRT et al. 2008]

R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, X. Zhu "No "Power" Struggles: Coordinated Multi-level Power Management for the Data Center" New York, NY, USA, Heft 1. 10.1145/1353534.1346289 ISSN: 0163-5964, 2008.

[Rum 1991]

J. Rumbaugh "Object oriented modeling and design". [Hauptbd.]. 0-13-629841-9, 1991.

[RZB et al. 2012]

Rongliang Zhou, Zhikui Wang, C. Bash, A. McReynolds "Data center cooling management and analysis - a model based approach" Semiconductor Thermal Measurement and Management Symposium (SEMI-THERM), 2012 28th Annual IEEE. 10.1109/STHERM.2012.6188832 Print ISBN:978-1-4673-1110-6, 2012.

[SA 2003]

J. Srinivasan, S. Adve "Predictive Dynamic Thermal Management for Multimedia Applications" Proceedings of the 17th Annual International Conference on Supercomputing. 10.1145/782814.782831 ISBN:1-58113-733-8, 2003.

[Sah 2011]

Saha Sonal "An Experimental Evaluation of Real-Time DVFS Scheduling Algorithms" Blacksburg, Virginia, 2011.

[SAJ et al. 2010]

Sverre Jarp, Alfio Lazzaro, Julien Leduc, Andrzej Nowak "Evaluation of the Intel Nehalem-EX server processor" CERN openlab, 2010.

[SAS 2002]

K. Skadron, T. Abdelzaher, M. Stan "Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management" High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium on. 10.1109/HPCA.2002.995695 ISSN :1530-0897, 2002.

[SBA et al. 2011]

Song Liu, Brian Leung, Alexander Neckar, Seda Ogrenci Memik, Gokhan Memik, Nikos Hardavellas "Hardware/software techniques for DRAM thermal management" In Proceedings of the 17th IEEE International Symposium on High Performance Computer Architecture (HPCA), 2011, 2011.

[SBM 2009]

K. Singh, M. Bhadauria, S. McKee "Real Time Power Estimation and Thread Scheduling via Performance Counters" SIGARCH Comput. Archit. News. 37, Heft 2. 10.1145/1577129.1577137, 2009.

[SHY et al. 2005]

Shuo Kang, Huayong Wang, Yu Chen, Xiaoge Wang, Yiqi Dai "Skyeye: An Instruction Simulator with Energy Awareness" Embedded Software and Systems. 10.1007/11535409_66 Print ISBN978-3-540-28128-3, 2005.

[SIC 2003]

A. Sinha, N. Ickes, A. Chandrakasan "Instruction level and operating system profiling for energy exposed software". 11, Heft 6. 10.1109/TVLSI.2003.819569 ISSN: 1063-8210, 2003.

[Sir 2007]
W. Siricharoen "Ontologies and Object models in Object Oriented Software Engineering". 33 IAENG International Journal of Computer Science, 2007.

[SKO et al. 1997]
H. Sanchez, B. Kuttanna, T. Olson, M. Alexander, G. Gerosa, R. Philip, J. Alvarez "Thermal management system for high performance PowerPC/sup TM/ microprocessors" Compcon '97. Proceedings, IEEE. 10.1109/CMPCON.1997.584744 ISSN :1063-6390, 1997.

[Sku 2013]
H. Skurk "Reliable Data Centre", Bitkom, 2013.

[SLU 2010]
S. Sivathanu, Ling Liu, C. Ungureanu "Modeling the performance and energy of storage arrays" Green Computing Conference, 2010 International. 10.1109/GREENCOMP.2010.5598308 Print ISBN:978-1-4244-7612-1, 2010.

[SM 2001]
A. Sangiovanni-Vincentelli, G. Martin "Platform-based design and software design methodology for embedded systems" IEEE Design Test of Computers. 18, Heft 6. 10.1109/54.970421 ISSN: 0740-7475, 2001.

[SMA et al. 2003]
Skadron, Martonosi, August, Hill, Lilja, Pai "Challenges in computer architecture evaluation" Computer. 36. 10.1109/MC.2003.1220579 ISSN :0018-9162, 2003.

[SPE 2015]
SPEC Standard Performance Evaluation Corporation "Fujitsu FUJITSU Server PRIMERGY RX2560 M1", 2015.

[SR 2012]
S. Saha, B. Ravindran "An Experimental Evaluation of Real-Time DVFS Scheduling Algorithms" SYSTOR '12 Proceedings of the 5th Annual International Systems and Storage. 5. 10.1145/2367589.2367604 ISBN: 978-1-4503-1448-0, 2012.

[SSG et al. 2009]

Seung-Hwan Lim, B. Sharma, Gunwoo Nam, Eun Kyoung Kim, C. Das "MDCSim: A multi-tier data center simulation, platform" Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on. 10.1109/CLUSTR.2009.5289159 Print ISBN:978-1-4244-5011-4, 2009.

[SSH 2014]

K. Siozios, D. Soudris, M. Hübner "A Framework for Supporting Adaptive Fault-Tolerant Solutions" New York, NY, USA. 13, Heft 5. 10.1145/2629473, 2014.

[SSS et al. 2004]

K. Skadron, M. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, D. Tarjan "Temperature-aware Microarchitecture: Modeling and Implementation" ACM Trans. Archit. Code Optim., Heft 1. 10.1145/980152.980157, 2004.

[Ste 2012]

R. Steinbrecher "IT Equipment Thermal Management and Controls" American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc., 2012.

[Stö 2014]

H. Stöcker "Taschenbuch der Physik". Verl. Europa-Lehrmittel Nourney, Vollmer Haan-Gruiten. 7. Aufl. 978-3-8085-5677-1, 2014.

[SWK 2011]

Y. Shi, S. Wang, G. Kou "New State of MCDM in the 21st Century". Springer-Verlag Berlin Heidelberg Berlin, Heidelberg, 2011.

[SXC et al. 2000]

Shiwei Feng, Xuesong Xie, Changzhi Lu, G. Shen, Guangbo Gao, Xiaoling Zhang "The thermal characterization of packaged semiconductor device" Semiconductor Thermal Measurement and Management Symposium, 2000. Sixteenth Annual IEEE. 10.1109/STHERM.2000.837087 Print ISBN:0-7803-5916-X, 2000.

[TDM 2011]

I. Takouna, W. Dawoud, C. Meinel "Accurate Mutlicore Processor Power Models for Power-Aware Resource Management" Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on. 10.1109/DASC.2011.85 Print ISBN:978-1-4673-0006-3, 2011.

[Tep 2010]

N. Tepper "Exploring the use of Model-Based Systems Engineering (MBSE) to develop systems architectures in naval ship design", 2010.

[THS 2010]

D. Tsirogiannis, S. Harizopoulos, M. Shah "Analyzing the Energy Efficiency of a Database Server" Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. 10.1145/1807167.1807194 ISBN: 978-1-4503-0032-2, 2010.

[Thy 2012]

Marc Thylmann, Bitkom, “Stromverbrauch von Rechenzentren und Servern sinkt”, Berlin, 21. Mai 2012, http://www.pressebox.de/pressemitteilung/bitkom-bundesverband-informationswirtschaft-telekommunikation-und-neue-medien-ev/Stromverbrauch-von-Rechenzentren-und-Servern-sinkt/boxid/509627

[TKS et al. 2013]

M. Tighe, G. Keller, J. Shamy, M. Bauer, H. Lutfiyya "Towards an improved data centre simulation with DCSim" Network and Service Management (CNSM), 2013 9th International Conference on. 10.1109/CNSM.2013.6727859, 2013.

[TMW 1994]

V. Tiwari, S. Malik, A. Wolfe "Power analysis of embedded software: a first step towards software power minimization" Very Large Scale Integration (VLSI) Systems, IEEE Transactions on. 2, Heft 4. 10.1109/92.335012 ISSN: 1063-8210, 1994.

[TMW et al. 1996]

V. Tiwari, S. Malik, A. Wolfe, M. Tien-Chien Lee "Instruction level power analysis and optimization of software" Journal of VLSI signal processing systems for signal, image and video technology. 13, 2-3. 10.1007/BF01130407 Print ISSN: 0922-5773, 1996.

[Tol 2009]
M. Tolentino "Managing Memory for Power, Performance, and Thermal Efficiency, PhD thesis", 2009.

[TRJ 2005]
T. Tan, A. Raghunathan, N. Jha "Energy Macromodeling of Embedded Operating Systems" ACM Transactions on Embedded Computing Systems (TECS). 4, Heft 1. 10.1145/1053271.1053281, 2005.

[TSK 2009]
P.-N. Tan, M. Steinbach, V. Kumar "Introduction to data mining". Boston [u.a.] : Pearson, Addison Wesley Boston. [Nachdr.]. 0-321-32136-7, 2009.

[TSW 2009]
Thanh Do, Suhib Rawshdeh, Weisong Shi "ptop: A process-level power profiling tool" in Proceedings of the 2nd Workshop on Power Aware Computing and Systems (HotPower'09},, 2009.

[TSX et al. 2003]
Tschudi, Sreedharan, Xu, Coup, Roggensack "Data Centers and Energy Use – Lets Look at the Data", 2003.

[UKI et al. 2010]
R. Urgaonkar, U. Kozat, K. Igarashi, M. Neely "Dynamic resource allocation and power management in virtualized data centers" Network Operations and Management Symposium (NOMS), 2010 IEEE. 10.1109/NOMS.2010.5488484 ISSN :1542-1201, 2010.

[Wal 2007]
Wallis "A Beginner's Guide to Graph Theory". Birkhäuser Boston. 2. 978-0-8176-4484-0, 2007.

[WB 2004]
A. Weissel, F. Bellosa "Dynamic Thermal Management for Distributed Systems" In proceedings of the first workshop on temperature-aware computer systems (TACS'04), 2004.

[WJ 1996]

S. Wilton, N. Jouppi "CACTI: an enhanced cache access and cycle time model" Solid-State Circuits, IEEE Journal of. 31, Heft 5. 10.1109/4.509850 ISSN :0018-9200, 1996.

[WK 2013]

L. Wang, S. Khan "Review of Performance Metrics for Green Data Centers: A Taxonomy Study" Hingham, MA, USA, Heft 3. 10.1007/s11227-011-0704-3, 2013.

[WWF et al. 2003]

R. Wunderlich, T. Wenisch, B. Falsafi, J. Hoe "SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling" Proceedings of the 30th Annual International Symposium on Computer Architecture, Heft 2. 10.1145/871656.859629 ISBN:0-7695-1945-8, 2003.

[WWF et al. 2006]

T. Wenisch, R. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, J. Hoe "SimFlex: Statistical Sampling of Computer System Simulation" Micro, IEEE. 26, Heft 4. 10.1109/MM.2006.79 ISSN :0272-1732, 2006.

[XMM 2003]

F. Xie, M. Martonosi, S. Malik "Compile-time Dynamic Voltage Scaling Settings: Opportunities and Limits" Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation. 38, Heft 5. 10.1145/780822.781138 ISBN:1-58113-662-5, 2003.

[XTB 2007]

Xiaoyao Liang, K. Turgay, D. Brooks "Architectural power models for sram and cam structures based on hybrid analytical/empirical techniques" Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on. 10.1109/ICCAD.2007.4397367, 2007.

[YP 2009]

Yu Bai, Priya Vaidya "MEMORY CHARACTERIZATION TO ANALYZE AND PREDICT MULTIMEDIA PERFORMANCE AND POWER IN EMBEDDED SYSTEMS" Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on. 10.1109/ICASSP.2009.4959835, 2009.

[YS 2005]
Yong Zhan, S. Sapatnekar "A high efficiency full-chip thermal simulation algorithm" Computer-Aided Design, 2005. ICCAD-2005. IEEE/ACM International Conference on. 10.1109/ICCAD.2005.1560144 Print ISBN:0-7803-9254-X, 2005.

[YSY et al. 2011]
Younghyun Kim, Sangyoung Park, Youngjin Cho, Naehyuck Chang "System-Level Online Power Estimation Using an On-Chip Bus Performance Monitoring Unit" Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions o, Heft 11. 10.1109/TCAD.2011.2160349, 2011.

[YVK et al. 2000]
W. Ye, N. Vijaykrishnan, M. Kandemir, M. Irwin "The Design and Use of Simplepower: A Cycle-accurate Energy Estimation Tool" DAC '00 Proceedings of the 37th Annual Design Automation Conference. 10.1145/337292.337436 ISBN:1-58113-187-9, 2000.

[YZ 2011]
Yao Yingbiao, Zeng Xianbin "Fast, Accurate On-Chip Data Memory Performance Estimation" Computational Science and Engineering (CSE), 2011 IEEE 14th International Conference on. 10.1109/CSE.2011.19, 2011.

[ZMC 2003]
D. Zhu, R. Melhem, B. Childers "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems" IEEE Transactions on Parallel and Distributed Systems. 14, Heft 7. 10.1109/TPDS.2003.1214320 ISSN=1045-9219, 2003.

[ZT 2011]
Zoltan Majo, Thomas R. Gross "Memory System Performance in a NUMA Multicore Multiprocessor" SYSTOR '11 Proceedings of the 4th Annual International Conference on Systems and Storage. 4. 10.1145/1987816.1987832, 2011.

[ZX 2012]
J. Zhao, Y. Xie "Optimizing bandwidth and power of graphics memory with hybrid memory technologies and adaptive data migration" ISSN 1092-3152, 2012.

[ZYY et al. 2011]

Zehan Cui, Yan Zhu, Yungang Bao, Mingyu Chen "A fine-grained component-level power measurement method" Green Computing Conference and Workshops (IGCC), 2011 International. 10.1109/IGCC.2011.6008599 Print ISBN:978-1-4577-1222-7, 2011.

[ZZX 2000]

Z. Zhang, Z. Zhu, Z. Xiaodong "A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality". 10.1109/MICRO.2000.898056 ISSN 1072-4451, 2000.

## A7. Notes