Linköping Studies in Science and Technology. Dissertations, No. 1887

System-Level Analysis and Design under Uncertainty

Ivan Ukhov

Department of Computer and Information Science
Linköping University
SE-581 83 Linköping, Sweden

Linköping 2017

© 2017 Ivan Ukhov

ISBN 978-91-7685-426-6
ISSN 0345-7524
URL https://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-140758

Printed by LiU-Tryck, Linköping 2017

To my parents,
Tatiana and Aleksandr

Abstract

One major problem for the designer of electronic systems is the presence of uncertainty, which is due to phenomena such as process and workload variation. Very often, uncertainty is inherent and inevitable. If ignored, it can lead to degradation of the quality of service in the best case and to severe faults or burnt silicon in the worst case. Thus, it is crucial to analyze uncertainty and to mitigate its damaging consequences by designing electronic systems in such a way that uncertainty is effectively and efficiently taken into account.

We begin by considering techniques for deterministic system-level analysis and design of certain aspects of electronic systems. These techniques do not take uncertainty into account, but they serve as a solid foundation for those that do. Our attention revolves primarily around power and temperature, as they are of central importance for attaining robustness and energy efficiency. We develop a novel approach to dynamic steady-state temperature analysis of electronic systems and apply it in the context of reliability optimization.

We then proceed to develop techniques that address uncertainty. The first technique is designed to quantify the variability in process parameters, which is induced by process variation, across silicon wafers based on indirect and potentially incomplete and noisy measurements. The second technique is designed to study diverse system-level characteristics with respect to the variability originating from process variation. In particular, it allows for analyzing transient temperature profiles as well as dynamic steady-state temperature profiles of electronic systems. This is illustrated by considering a problem of design-space exploration with probabilistic constraints related to reliability. The third technique that we develop is designed to efficiently tackle the case of sources of uncertainty that are less regular than process variation, such as workload variation. This technique is exemplified by analyzing the effect that workload units with uncertain processing times have on the timing-, power-, and temperature-related characteristics of the system under consideration.

We also address the issue of runtime management of electronic systems that are subject to uncertainty. In this context, we perform an early investigation into the utility of advanced prediction techniques for the purpose of fine-grained long-range forecasting of resource usage in large computer systems.

All the proposed techniques are assessed by extensive experimental evaluations, which demonstrate the superior performance of our approaches to analysis and design of electronic systems compared to existing techniques.

*The research presented in this thesis has been partially funded by the National Computer Science Graduate School (*CUGS*) in Sweden.*

Sammanfattning

Ett stort problem för designern inom elektroniska system är förekomsten av osäkerhet, som beror på sådana fenomen som variationer relaterade till tillverkning och arbetsbelastning. Osäkerhet är i många fall naturlig och oundviklig och kan leda till försämring av servicekvaliteten i bästa fall och till allvarliga fel eller bränd kisel i värsta fall. Därför är det viktigt att analysera osäkerhet och att mildra dess skadliga följder genom att designa elektroniska system på sådant sätt att de effektivt och ändamålsenligt tar hänsyn till osäkerhet.

Vi börjar med att överväga tekniker för deterministisk systemnivåanalys och systemnivådesign av elektroniska system. Dessa tekniker tar inte hänsyn till osäkerhet; de fungerar dock som en solid grund för de tekniker som gör det. Vi fokuserar främst på faktorer som kraft och temperatur eftersom de är av central betydelse för att uppnå robusthet och energieffektivitet. Vi utvecklar ett nytt tillvägagångssätt för dynamisk stabiliserad temperaturanalys av elektroniska system och tillämpar det inom ramen för tillförlitlighetsoptimering.

Vi fortsätter sedan med att utveckla ett antal tekniker som tar hänsyn till osäkerhet i elektroniska system. Den första tekniken är utformad för att kvantifiera föränderligheten hos processparametrar, som framkallas av processvariation, över kiselplattor baserat på indirekta och potentiellt ofullständiga och bullriga mätningar. Den andra tekniken går ut på att studera olika systemnivåegenskaper med avseende på variabiliteten som härrör från processvariation. I synnerhet tillåter den analys av övergående temperaturprofiler samt dynamiska stabiliserade temperaturprofiler hos elektroniska system. Detta illustreras genom att överväga problemet med utforskning av ett designutrymme med probabilistiska begränsningar relaterade till tillförlitlighet. Den tredje tekniken som vi utvecklar är utformad för att effektivt ta itu med osäkerhetsfaktorer som är mindre regelbundna än processvariation, som till exempel variationer i arbetsbelastning. Denna teknik exemplifieras genom att analysera den effekt som arbetsbelastningsenheter med osäkra behandlingstider har på det aktuella systemets tids-, kraft- och temperaturrelaterade egenskaper.

Vi tar även hänsyn till frågan om körtidshantering av elektroniska system under osäkerhet. I det här sammanhanget utför vi en tidig undersökning av användbarheten av avancerade prediktiva tekniker för att anskaffa en förfinad prognos för långsiktig användning av resurser i stora datorsystem.

Alla föreslagna tekniker bedöms genom omfattande experimentella utvärderingar som påvisar den överlägsna prestationsförmågan av våra metoder för analys och design av elektroniska system med avseende på befintliga tekniker.

Acknowledgments

I would like to express my enormous gratitude to Professor Zebo Peng and Professor Petru Eles for their exhaustive and discerning supervision of my graduate studies at Linköping University. I cannot possibly imagine a better working environment than the one that Zebo and Petru have carefully created and maintained over all these years in our Embedded Systems Laboratory.

I am extremely grateful to Professor Diana Marculescu for her caring and thorough supervision during my visit to Carnegie Mellon University. Being part of a different research group in a different country for several months was an invaluable professional experience for me, and Diana made sure that this experience was also impeccably smooth and indispensably delightful.

I would like to thank Professor Mattias Villani for his valuable advice and constant readiness to lend a natural-born Bayesian's helping hand to those of us who occasionally or always find ourselves hopelessly lost in statistics.

I am thankful to Anne Moe for attentively shepherding graduate students toward doctoral hats, which must be akin in difficulty to herding cats. I am also thankful to Marie Johansson for her kind help with every matter.

Over the course of my doctoral education, I have had the pleasure to meet many other amazing people who have been instrumental in making it possible for me to reach the finish line of this challenging and gratifying journey.

I thank Sergiu Rafiliu for welcoming me at the train station on my very first day in Linköping. I thank Min Bao for helping me take my very first steps as a researcher. I thank Soheil Samii for being an excellent example of an accomplished senior graduate student, which I aspired to become myself one day.

I am grateful to Breeta SenGupta for sharing my passion for drawing and photography. I am grateful to Unmesh Bordoloi and Ahmed Rezine for being supportive and reassuring, which I really needed at times. I am also grateful to Bogdan Tanasa for many engaging discussions at the whiteboard.

I am thankful to Maria Vasilevskaya for her constantly positive attitude and contagious optimism. I am also thankful to Lisa Malmberg, Erik Hansson, and Ulf Kargén for patiently answering my myriad of questions about Swedish.

I am particularly grateful to Flavia Horga, Adrian Horga, Sarah Alsaadi, Adrian Lifa, Mikaela Holmbäck, Arian Maghazeh, and Antonia Arvanitaki for our numerous merry gatherings both on and off the university's campus.

Lastly and most importantly, I cannot thank my parents, Tatiana Ukhova and Aleksandr Ukhov, enough. Their unconditional and inexhaustible care and support have always been of paramount significance to me.

Ivan Ukhov
Linköping, November 2017

Table of Contents

List of Algorithms

List of Figures

List of Tables

1 Introduction

Electronic systems are omnipresent and omniscient. They continue to invade and conquer new application areas, penetrating deep into everyday life and becoming unsettlingly indispensable for the tasks entrusted to them. However, there is a great deal of uncertainty associated with electronic systems, and as they continue to flourish, the impact of this uncertainty inevitably becomes more prominent and entails more severe consequences, requiring appropriate treatment. It is then readily understandable that analysis and design of electronic systems are intensely difficult and vastly far-reaching endeavors.

Analysis and design of electronic systems with an emphasis on uncertainty is the topic of this thesis. In this section, we give an introduction to our work and, in particular, summarize our motivations, objectives, and contributions. However, in order to give a clear presentation, we first elaborate on what is meant by the word *uncertainty* in the context of the thesis.

1.1 Uncertainty

In this thesis, we take the designer's standpoint. Specifically, a particular aspect of the electronic system under consideration is said to be uncertain if this aspect is unknown to the designer of the system at design time.

The typical scenario is that the designer is interested in studying a certain quantity—referred to as the quantity of interest—where complete knowledge of this quantity would be highly beneficial but cannot be obtained, since the quantity *per se* is uncertain to the designer, or because it depends on parameters that are uncertain to the designer. We remain agnostic on the underlying reason behind this state of uncertainty: it can be aleatory as well as epistemic.

Uncertainty in electronic systems can originate from different phenomena, and, in many cases, it is inherent and inevitable (from the designer's standpoint). Let us consider three examples that are of relevance to the thesis.

1.1.1 Process Variation

A prominent example of uncertainty is the one that stems from process variation [12, 103]. In this case, the source of uncertainty is the fabrication process. Specifically, the process parameters of fabricated nanoscale devices deviate from their nominal values, since the contemporary fabrication process cannot be controlled precisely down to the level of individual atoms.

The aforementioned transistor-level variability propagates to such crucial system-level characteristics of an electronic system as power consumption and heat dissipation, thereby making them uncertain to the designer of this system. The propagation is due to process variation affecting the key parameters of a technological process, such as the effective channel length and gate oxide thickness. As a result, the same workload applied to two seemingly identical dies can lead to two drastically different power profiles and consequently to two drastically different temperature profiles, since power consumption and heat dissipation depend on the aforementioned quantities, which will be discussed further in Section 2.2. This concern is especially exigent due to the power-temperature interplay—which is also covered in Section 2.2 as well as in Section 3.5—whose magnitude depends on process parameters.

1.1.2 Workload Variation

Another salient example of uncertainty is the one that emerges from workload variation. In this case, the source of uncertainty is the actual work that electronic systems are instructed to perform. To elaborate, from one activation to another, the same piece of deterministic software can exhibit drastically different behaviors depending on the environment and input data. Neither the environment nor the input data that the system under consideration will be exposed to at runtime is exhaustively known at early development stages.

1.1.3 Aging Variation

Yet another example of uncertainty is the one that originates from aging variation. In this case, the source of uncertainty is natural or accelerated wear [37], which leads to the degradation over time of the performance of electrical circuits. This degradation can cause terminal faults and, therefore, can abruptly end the life of the system at hand. Since the degradation is a nonuniform and intricate process, the lifetime of the system is uncertain to the designer.

There is one factor that is worth highlighting in this context: temperature. Temperature has a profound impact on the lifetime of electronic systems,

which is well known and well studied. The failure mechanisms that are commonly considered—such as electromigration, time-dependent dielectric breakdown, and thermal cycling—are directly driven by temperature. Additionally, there is a connection between process variation and aging variation, which is that the former intensifies the latter via temperature.

Having introduced and exemplified uncertainty in electronic systems, we are ready to consolidate our motivation and solidify our objective.

1.2 Motivation

In the context of current and future technologies, it is both inefficient and unreliable to rely exclusively on uncertainty-unaware techniques when developing electronic systems. Regardless of its origin, uncertainty causes certain aspects of the system under consideration to be nondeterministic to the designer. Therefore, if the presence of uncertainty is disregarded, it can lead to degradation of the quality of service in the best case and to severe faults or burnt silicon in the worst case. Under these circumstances, it is crucial to analyze and quantify uncertainty in electronic systems and to mitigate its deleterious implications by designing electronic systems in such a way that uncertainty is taken into consideration in an effective and efficient manner.

Even in the deterministic scenario, analysis and design of electronic systems are difficult undertakings. However, uncertainty exacerbates the situation even further, requiring adequate techniques for analysis and design that allow for uncertainty. Versatile uncertainty-aware tools are meager at best, and in the case of certain problems, they are non-existent. However, such techniques are a must; they are an indispensable asset of design workflows that strive to provide guarantees on efficiency and robustness for their products.

1.3 Objective

Our goal is to assist the designer of electronic systems by providing effective and efficient system-level tools for analysis and design under uncertainty. To this end, we are to develop techniques for quantifying and mitigating the variability that originates from the fabrication process, workload, and aging.

In particular, we intend to develop solutions that are applicable to diverse practical problems and that are flexible and straightforward to use. In addition, we are interested in calculating probability distributions—which are exhaustive characterizations—rather than, for instance, corner cases, since designing for the worst case often leads to a conservative and overdesigned system.

1.4 Contribution

Our work has made the following major contributions:

- We have proposed a fast and accurate approach to deterministic dynamic steady-state temperature analysis and have improved deterministic transient temperature analysis. Leveraging our efficient approach, we have developed a procedure for reliability optimization targeted at reducing thermal-cycling fatigue and hence at alleviating aging uncertainty.
- We have developed a versatile statistical framework for analyzing the variability in process parameters across silicon wafers that is induced by the fabrication process. The key feature of our approach is that it uses indirect measurements, which streamlines its usage and allows for a reduction in the costs associated with the production process.
- We have developed a versatile probabilistic framework for analyzing system-level quantities that are affected by process variation. Using our technique, we have enhanced reliability analysis and optimization of electronic systems by accounting for process uncertainty in reliability models, which has also improved the treatment of aging uncertainty.
- We have also developed a versatile probabilistic framework for analyzing system-level quantities affected by workload variation, which tends to engender a less regular variability than the one that stems from the fabrication process and thus necessitates a different technique.
- We have performed an early yet informative investigation into the utility of the latest advancements in machine learning for mitigating workload uncertainty at runtime in the context of resource management. Specifically, we have applied advanced prediction techniques to fine-grained long-range forecasting of resource usage in large computer systems.

All the proposed solutions have been assessed by extensive experiments, which have demonstrated the superior performance of our approaches to analysis and design of electronic systems compared to existing techniques.

1.5 Previous Work

Since the appearance of the first digital computers in the 1940s, Monte Carlo (MC) sampling remains one of the most well-known and widely used methods for analyzing stochastic systems. With this technique, the system at hand is treated as an opaque object, and one only has to evaluate this object a number of times in order to start to draw conclusions about the system's behavior. This straightforwardness of application is at the heart of the technique's popularity, but there are two other reasons: the independence of the number of stochastic

dimensions and the law of large numbers [35], which states that the quantities estimated using MC sampling asymptotically approach the true values.

The major problem with sampling-based methods, however, is in sampling *per se*: one has to obtain a sufficiently large number of readings of the quantity of interest in order to accurately estimate the necessary statistics about this quantity, and the number of required samples can be considerable [33]. In the case of MC sampling, for instance, the error can be halved by quadrupling the number of sample points. In other words, an additional decimal point of accuracy requires a hundred times more samples. When the subject of the analysis is computationally expensive—which is arguably the case with all non-trivial quantities of interest, as they typically involve an evaluation of the whole system—sampling methods are rendered slow and often unfeasible.

There are sampling techniques that have better convergence rates than that of classical MC sampling. Examples of such techniques include quasi-MC sampling and Latin hypercube sampling [2, 57]. However, these convergence rates are still relatively low, and the corresponding techniques often have additional restrictions, which limit their applicability in practice [120].

In order to circumvent the high computational costs associated with sampling methods and to address other scenarios, a number of alternative techniques have been developed for uncertainty-aware analysis and design of electronic systems. Process variation has been the topic of many lines of research; see, for instance, [6, 7, 11, 58, 69, 117]. Similarly, workload uncertainty has not been deprived of attention; see, for instance, [32, 88, 96, 98, 105, 124]. Aging uncertainty has also been studied extensively in the literature; see, for instance, [24, 28, 39, 50, 61, 83]. However, certain important problems have not been addressed yet, and in the case of the ones that have been considered, the proposed solutions are often restricted in use, which is due in part to the unrealistic assumptions that these solutions make. The aforementioned concerns will be discussed in detail in the relevant parts of the thesis.

1.6 Thesis Overview

The following is a bird's eye view of the remaining chapters of the thesis.

In Chapter 2, we introduce a number of commonly used models that constitute a starting point for the developments presented in this thesis. Specifically, we elaborate on a general system model, a temperature-aware power model, a temperature model, and a temperature-aware reliability model.

In Chapter 3, we consider techniques for deterministic system-level analysis of electronic systems. These techniques do not take uncertainty into account; however, they serve as a solid foundation for those that do. Our attention revolves primarily around power and temperature, since these two factors are of central importance for attaining robustness and energy efficiency. We develop a novel approach to dynamic steady-state temperature analysis of elec-

tronic systems and apply it in the context of temperature-aware reliability optimization. Reliability optimization addresses aging uncertainty by definition; however, this optimization falls within the scope of this chapter, since the accompanying reliability analysis treats temperature as a deterministic quantity, which is suboptimal, and which we address in the subsequent chapters.

In Chapter 4, we present our first technique that is targeted at uncertainty. We develop a statistical approach to analyzing process-variation-induced fluctuations in process parameters across silicon wafers by means of indirect measurements, which can also be incomplete and noisy. Examples of process parameters include the effective channel length and gate oxide thickness, and examples of indirect measurements include readings from thermal sensors. To this end, we make use of a suite of tools borrowed from Bayesian statistics.

In Chapter 5, we continue working with process uncertainty and present a technique that is applicable to studying diverse quantities with respect to process variation. To this end, we leverage the theory of polynomial chaos expansions. In particular, the proposed approach allows for analyzing transient and dynamic steady-state power and temperature profiles of electronic systems as well as such critical metrics of electronic systems as their peak temperatures and energy consumption. All the above quantities can be analyzed from a probabilistic perspective, and the utility of this is illustrated by addressing a problem of design-space exploration with a set of probabilistic constraints that are related to reliability. Unlike the reliability model utilized in Chapter 3, the one presented in Chapter 5 takes process uncertainty into consideration and consequently allows for a more adequate treatment of aging uncertainty.

In Chapter 6, we develop another system-level technique that is targeted at uncertainty. The tool leverages adaptive hierarchical interpolation on sparse grids and specializes in tackling workload variation and similar phenomena whose manifestation is less regular than the smooth and well-behaved one of process variation. The proposed technique is exemplified by quantifying the effect that workload units with unknown processing times have on the timing-, power-, and temperature-related characteristics of electronic systems.

In Chapter 7, we elaborate on runtime management of electronic systems under workload variation. In this context, we perform an early investigation into the applicability of advanced prediction techniques from machine learning to fine-grained long-range forecasting of resource usage in computer systems. Concretely, we study the applicability of recurrent neural networks.

In Chapter 8, we conclude the thesis by summarizing the main outcomes of our work and elaborating on potential directions for further development.

The thesis also contains an appendix in which we give an overview of a number of concepts from such interconnected disciplines as linear algebra, probability theory, statistics, numerical integration, and interpolation. The theory that is discussed in the appendix is utilized extensively throughout the thesis.

1.7 Publication Overview

The content of the thesis is chiefly based on the following two conference papers and three journal articles (in the order used in the bibliography):

[110] I. Ukhov, M. Bao, P. Eles, and Z. Peng. "Steady-state dynamic temperature analysis and reliability optimization for embedded multiprocessor systems." In: *Design Automation Conference*. June 2012, pp. 197–204. DOI: 10.1145/2228360.2228399.

[111] I. Ukhov, P. Eles, and Z. Peng. "Probabilistic analysis of power and temperature under process variation for electronic system design." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33.6 (June 2014), pp. 931–944. DOI: 10.1109/TCAD.2014.2301672.

[112] I. Ukhov, P. Eles, and Z. Peng. "Temperature-centric reliability analysis and optimization of electronic systems under process variation." In: *IEEE Transactions on Very Large Scale Integration Systems* 23.11 (Nov. 2015), pp. 2417–2430. DOI: 10.1109/TVLSI.2014.2371249.

[113] I. Ukhov, P. Eles, and Z. Peng. "Probabilistic analysis of electronic systems via adaptive hierarchical interpolation." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36.11 (Nov. 2017), pp. 1883–1896. DOI: 10.1109/TCAD.2017.2705117.

[116] I. Ukhov, M. Villani, P. Eles, and Z. Peng. "Statistical analysis of process variation based on indirect measurements for electronic system design." In: *Asia and South Pacific Design Automation Conference*. Jan. 2014, pp. 436–442. DOI: 10.1109/ASPDAC.2014.6742930.

In addition, the following two technical reports are used in the thesis:

[114] I. Ukhov, D. Marculescu, P. Eles, and Z. Peng. *Fast synthesis of power and temperature profiles for the development of data-driven resource managers*. Tech. rep. Linköping University, Sept. 2017.

[115] I. Ukhov, D. Marculescu, P. Eles, and Z. Peng. *Fine-grained long-range prediction of resource usage in computer clusters*. Tech. rep. Linköping University, Sept. 2017.

The publications listed above correspond to the chapters of the thesis as follows. The work on uncertainty-unaware analysis and design presented in [110] is covered in Chapter 3. The approach to characterization of process variation developed in [116] is presented in Chapter 4. The techniques for analysis and design under process variation proposed in [111] and [112] are amalgamated in Chapter 5. The approach to analysis under workload variation introduced in [113] is elaborated on in Chapter 6. The work on workload

prediction for resource management given in [115] is described in Chapter 7, which is also closely related to the data synthesis discussed in [114].

Lastly, the following two conference papers are not discussed in the thesis, but they are closely related to the subject of Chapter 7:

[80] M. Niknafs, I. Ukhov, P. Eles, and Z. Peng. "Two-phase interarrival time prediction for runtime resource management." In: *Euromicro Conference on Digital System Design*. Aug. 2017, pp. 524–528.

[81] M. Niknafs, I. Ukhov, P. Eles, and Z. Peng. "Workload prediction for runtime resource management." In: *IEEE Nordic Circuits and System Conference*. Oct. 2017.

2 Background

In this chapter, we present a number of common system-level models that the discussions in the subsequent chapters are based on.

2.1 System Model

Consider a generic electronic system that consists of n_p heterogeneous processing elements and is equipped with a thermal package. A processing element is any active component of the system, that is, any component that consumes power and dissipates heat. The thermal package is the cooling equipment of the system, including all passive components, that is, all components that do not consume power. Processing elements can be identified at different levels of granularity. For instance, a processing element could be a system on a chip, a central processing unit, an arithmetic logic unit, cache memory, or a communication bus. In the majority of the discussions in the thesis, processing elements correspond to processing units or cores of a multiprocessor system.

The system is characterized by power profiles and temperature profiles, which are discrete representations of the system's power consumption and heat dissipation, respectively. A power profile is an $n_p \times n_s$ matrix

$$\mathbf{P} = \left(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{n_s}\right) = \left(\boldsymbol{p}(t_1), \ldots, \boldsymbol{p}(t_{n_s})\right) \in \mathbb{R}^{n_p \times n_s} \tag{2.1}$$

that contains n_s samples $\{\boldsymbol{p}_i\}_{i=1}^{n_s} \subset \mathbb{R}^{n_p}$ of the power consumption of n_p processing elements taken at n_s moments in time $\{t_i\}_{i=1}^{n_s}$ with a sampling interval Δt such that $t_i - t_{i-1} = \Delta t$ for $i = 2, \ldots, n_s$. The number n_s is called the number of steps. Similarly, a temperature profile is an $n_p \times n_s$ matrix

$$\mathbf{Q} = \left(\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{n_s}\right) = \left(\boldsymbol{q}(t_1), \ldots, \boldsymbol{q}(t_{n_s})\right) \in \mathbb{R}^{n_p \times n_s} \tag{2.2}$$

that captures the heat dissipation of the processing elements at a number of equidistant moments in time. Note that, even though the exact time frame and

sampling interval of a power profile or a temperature profile are not included in the corresponding notation, they are understood from the context.

2.2 Power Model

The total power consumption of an electrical circuit consists of two main components: dynamic and static. Dynamic power is the result of the actual useful work done by the circuit. By contract, static power is the result of various parasitic currents that cannot be entirely eliminated. The total power consumed by n_p processing elements at time t is then written as

$$\boldsymbol{p}(t) = \boldsymbol{p}_{\text{dyn}}(t) + \boldsymbol{p}_{\text{stat}}(t) \tag{2.3}$$

where $\boldsymbol{p}_{\text{dyn}} \in \mathbb{R}^{n_p}$ and $\boldsymbol{p}_{\text{stat}} \in \mathbb{R}^{n_p}$ are the aforementioned dynamic and static components of the total dissipation of power, respectively.

Dynamic power is modeled as follows:

$$p_{\text{dyn}} = CfV^2$$

where C is the effective switched capacitance, f is the frequency, and V is the supply voltage of the circuit. Static power is due chiefly to the leakage current [12, 58, 59, 103], especially in modern CMOS transistors. Leakage power and, by extension, static power are modeled as follows [73]:

$$p_{\text{stat}} = n_g V I_0 \left(a\, q^2 e^{\frac{\alpha V + \beta}{q}} + b\, e^{\gamma V + \delta} \right) \tag{2.4}$$

where q is the current temperature, V is the current supply voltage, I_0 is the average leakage current at the reference temperature and reference supply voltage, and n_g is the number of gates in the circuit. The quantities a, b, α, β, γ, and δ are technology-dependent constants, which can be found in [73]. There are two aspects to note with respect to the equations given above.

First, the dynamic power component does not depend on the operating temperature, whereas the static one does. The dependence of static (and hence total) power on temperature is positive and strong: a higher level of heat dissipation leads to a considerably higher level of power consumption. On the other hand, as more power is consumed, more heat is dissipated. Consequently, power and temperature constantly instigate each other; they are interdependent. This phenomenon cannot be neglected, since it leads to a higher level of power (energy) consumption than expected and a higher level of heat dissipation than expected and, in the worst case, could cause a thermal runaway.

Second, even though it is not explicitly spelled out, static (and hence total) power depends on a number of process parameters, including the effective channel length and gate oxide thickness. In particular, the effective channel length has one of the strongest impacts on the leakage current [58].

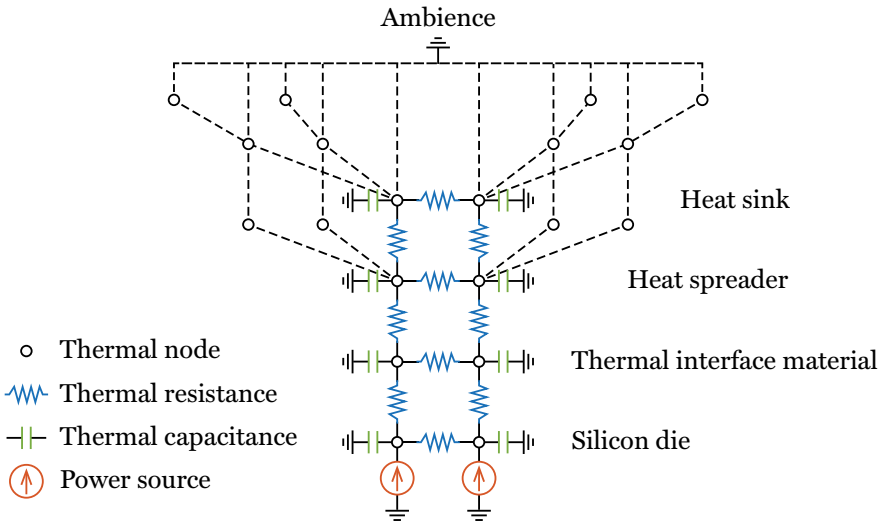

Figure 2.1: Example of a thermal RC circuit for a dual-core platform

2.3 Temperature Model

System-level temperature analysis of electronic systems is chiefly based on the duality between the transfer of heat and the transfer of electric charge [66]. The core idea is to construct a thermal RC circuit for the system under consideration [100]. Such a circuit is a collection of thermal nodes that are characterized by thermal capacitances and connected with each other via thermal resistances. The circuit is to capture the relevant physical structure and thermal properties of the system and thereby model its thermal dynamics.

Similarly to the identification of processing elements described in Section 2.1, a thermal RC circuit can be constructed at different levels of granularity, which, in this case, refers primarily to the number of thermal nodes denoted by n_n and their placement. It should be noted that the chosen level of granularity impacts the accuracy of the subsequent temperature calculations.

In order to give a better sense of thermal RC circuits, Figure 2.1 depicts a simplified example of such a circuit constructed for a dual-core chip that is equipped with a thermal package composed of a thermal interface material, heat spreader, and heat sink. Given n_p processing elements, each one is represented by one thermal node, and the thermal interface material, heat spreader, and heat sink are captured by n_p, n_p+4, and n_p+8 thermal nodes, respectively. Consequently, $n_n = 4 \times n_p + 12$. In this particular example, since $n_p = 2$, $n_n = 20$. The construction principle described in this paragraph is the one presented in [52], and it is also the one utilized throughout the thesis.

Suppose now that an adequate thermal RC circuit has been constructed for the electronic system under consideration. Regardless of the structure of the circuit, the thermal dynamics of the circuit, and thus of the electronic system itself, are governed by the following system of n_n differential and n_p algebraic equations, which is based on Fourier's law [40]:

$$\begin{cases} \mathbf{C}\dfrac{\mathrm{d}\tilde{\boldsymbol{s}}(t)}{\mathrm{d}t} + \mathbf{G}\tilde{\boldsymbol{s}}(t) = \tilde{\mathbf{B}}\boldsymbol{p}(t) & \text{(2.5a)} \\ \boldsymbol{q}(t) = \tilde{\mathbf{B}}^{\mathrm{T}}\tilde{\boldsymbol{s}}(t) + \boldsymbol{q}_{\mathrm{amb}} & \text{(2.5b)} \end{cases}$$

where $\boldsymbol{q} \in \mathbb{R}^{n_p}$, $\boldsymbol{q}_{\mathrm{amb}} \in \mathbb{R}^{n_p}$, and $\boldsymbol{p} \in \mathbb{R}^{n_p}$ are the operating temperature, ambient temperature, and power consumption of the processing elements, respectively. Furthermore, $\tilde{\boldsymbol{s}} \in \mathbb{R}^{n_n}$ represents the state of the thermal nodes, and $\mathbf{C} \in \mathbb{R}^{n_n \times n_n}$ and $\mathbf{G} \in \mathbb{R}^{n_n \times n_n}$ are a diagonal matrix of the thermal capacitance and a symmetric positive definite matrix of the thermal conductance, respectively. Lastly, $\tilde{\mathbf{B}} \in \mathbb{R}^{n_n \times n_p}$ is a mapping between the processing elements and the thermal nodes, which, without loss of generality, is assumed to be a rectangular diagonal matrix whose diagonal elements are equal to unity.

In general, the differential part in Equation 2.5a is nonlinear due to $\boldsymbol{p}$, since we do not make any assumptions about its structure; see also the discussion in Section 2.2. Therefore, there is no closed-form solution to the system.

Lastly, let us note that temperature analysis of electronic systems is generally considered to be a computationally expensive operation. The long computation times can be severely limiting, especially when the analysis is to be performed many times for such purposes as design-space exploration. Thus, there is always a need for development of more efficient techniques.

2.4 Reliability Model

In this section, we turn our attention to temperature-aware reliability analysis.

Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space (see Appendix A.2), and let $L : \Omega \to \mathbb{R}$ be a random variable that represents the lifetime of the system. The lifetime is the time until the system experiences a fault after which it no longer meets certain requirements. Also, let $F(\cdot|\boldsymbol{g})$ be the distribution function of L, which gives the probability of failure before a certain moment in time, where $\boldsymbol{g}$ is a vector of parameters. The expectation $\mathbb{E}L$ is called the mean time to failure (MTTF). Lastly, the complementary distribution function of L is

$$R(t|\boldsymbol{g}) = 1 - F(t|\boldsymbol{g}),$$

which, in the context of reliability analysis, gives the probability of survival up to a certain moment in time and is called the reliability function of the system.

The lifetime L is a function of the lifetimes of the n_p processing elements that the system is composed of. Denote these individual lifetimes by $\{L_i : \Omega \to \mathbb{R}\}_{i=1}^{n_p}$. Each L_i is characterized by a physical model of wear [37] that describes

the stress that processing element i is exposed to. Each L_i is also assigned an individual $R_i(\cdot|\boldsymbol{g}_i)$, which models the failures due to this stress.

The structure of $R(\cdot|\boldsymbol{g})$ with respect to $\{R_i(\cdot|\boldsymbol{g}_i)\}_{i=1}^{n_p}$ is problem specific, and it can be especially diverse in the context of fault-tolerant systems. Therefore, $R(\cdot|\boldsymbol{g})$ is to be devised by the designer of the system under consideration. To give an example, suppose that the failure of any of the n_p processing elements makes the whole system fail, and that $\{L_i\}_{i=1}^{n_p}$ are conditionally independent given the parameters gathered in $\boldsymbol{g}$. In this scenario,

$$\begin{aligned} L &= \min_{i=1}^{n_p} L_i \text{ and} \\ R(t|\boldsymbol{g}) &= \prod_{i=1}^{n_p} R_i(t|\boldsymbol{g}_i). \end{aligned} \tag{2.6}$$

2.4.1 Periodic Thermal Stress

In this subsection, we present a model that is frequently used for characterizing the lifetime L_i of a single processing element when it is exposed to repetitive temperature-induced stress [50, 118]. The scenario under consideration is that the system is experiencing a periodic workload with a certain period, which is denoted by τ. The resulting temperature profile is a dynamic steady-state temperature profile, which will be discussed in Section 3.4.

The lifetime L_i is assumed to have a Weibull distribution as follows:

$$L_i|\boldsymbol{g}_i \sim \text{Weibull}(\eta_i, \beta_i)$$

where η_i and β_i are called the scale and shape parameters, respectively, and $\boldsymbol{g}_i = (\eta_i, \beta_i)$. The distribution function of L_i is

$$F_i(t|\boldsymbol{g}_i) = 1 - \exp\left(-\left(\frac{t}{\eta_i}\right)^{\beta_i}\right), \tag{2.7}$$

the reliability function of the processing element is

$$R_i(t|\boldsymbol{g}_i) = 1 - F_i(t|\boldsymbol{g}_i) = \exp\left(-\left(\frac{t}{\eta_i}\right)^{\beta_i}\right), \tag{2.8}$$

and the corresponding MTTF is

$$\mu_i = \mathbb{E}L_i = \eta_i \, \Gamma\left(1 + \frac{1}{\beta_i}\right) \tag{2.9}$$

where Γ stands for the gamma function.

Remark 2.1. *The Weibull model has a special case: assuming that $\beta_i = \beta$ for $i = 1, \ldots, n_p$, the reliability function in Equation 2.6 belongs to a Weibull distribution whose shape parameter is β and scale parameter is given by*

$$\eta = \left(\sum \left(\frac{1}{\eta_i} \right)^{\beta} \right)^{-\frac{1}{\beta}}.$$

It is natural to expect that the usage conditions and hence the corresponding stress change over the period τ. The distribution of L_i should then reflect this aspect, which is not prominent in Equation 2.7. In order to take this into account, the period is split into $n_{c,i}$ time intervals $\{\Delta t_{ij}\}_{j=1}^{n_{c,i}}$ so that the conditions that are relevant to the model remain unchanged in each interval. Let

$$L_{ij} | \boldsymbol{g}_{ij} \sim \text{Weibull}(\eta_{ij}, \beta_{ij})$$

be the time to failure that the processing element would have if interval j was the only interval present and denote the corresponding MTTF by μ_{ij}.

In the case of temperature-induced failures, we specify that $\beta_{ij} = \beta_i$ for $j = 1, \ldots, n_{c,i}$; that is, the shape parameters are all equal. The reason is that, unlike the scale parameters, the shape parameters are independent of the operating temperature [13]. As shown in [118], in this scenario, the reliability function of the processing element can still be approximated by means of Equation 2.8 with the following scale parameter:

$$\eta_i = \frac{\sum_{j=1}^{n_{c,i}} \Delta t_{ij}}{\sum_{j=1}^{n_{c,i}} \frac{\Delta t_{ij}}{\eta_{ij}}}.$$

Applying Equation 2.9 at the level of individual intervals, η_i is rewritten as

$$\eta_i = \frac{\sum_{j=1}^{n_{c,i}} \Delta t_{ij}}{\Gamma\left(1 + \frac{1}{\beta_i}\right) \sum_{j=1}^{n_{c,i}} \frac{\Delta t_{ij}}{\mu_{ij}}}.$$

Note now that the model in Equation 2.8 becomes fully specified as soon as Δt_{ij} and μ_{ij} are identified for $j = 1, \ldots, n_{c,i}$. This part depends on the particular failure mechanism that is being considered, which we discuss next.

2.4.2 Thermal-Cycling Fatigue

Let us tailor the above Weibull model to thermal-cycling fatigue [37]. This type of fatigue is of particular interest to us due to its prominent dependence on temperature fluctuations: apart from the average and maximum values, the frequencies and amplitudes of temperature oscillations matter in this case.

Time intervals with constant relevant conditions correspond to thermal cycles. In order to detect them in a given temperature curve, the curve is first

analyzed using a peak-detection algorithm in order to extract a set of extrema. The rainflow counting method [118] is then applied to these extrema. The result is a set of $n_{c,i}$ thermal cycles. Each detected cycle is characterized by a number of properties, including the desired duration Δt_{ij}. Regarding the corresponding μ_{ij}, it can be expressed as follows:

$$\mu_{ij} = n_{c,ij} \Delta t_{ij}$$

where $n_{c,ij}$ stands for the mean number of such cycles to failure.

Remark 2.2. *A cycle detected by the rainflow counting method does not have to be formed by adjacent extrema; cycles can overlap. This feature makes the counting method very efficient at mitigating overestimation. A cycle could be a half cycle, meaning that only an upward or downward swing is present in the time series, which is assumed to be adequately accounted for.*

The number $n_{c,ij}$ is estimated using a modified version of the Coffin–Manson equation with the Arrhenius term as follows [37, 118]:

$$n_{c,ij} = a_i (\Delta q_{ij} - \Delta q_{0,ij})^{-b_i} \exp\left(\frac{c_i}{k q_{\max,ij}}\right) \tag{2.10}$$

where a_i, b_i (called the Coffin–Manson exponent), and c_i (called the activation energy) are empirically determined constants; k is the Boltzmann constant; Δq_{ij} is the excursion of the cycle in question; $\Delta q_{0,ij}$ is the portion of the temperature excursion that resides in the elastic region, which does not cause damage; and $q_{\max,ij}$ is the maximum temperature during the cycle.

The reliability model of a single processing element is now fully specified. The reliability function is the one in Equation 2.8 with

$$\eta_i = \frac{\sum_{j=1}^{n_{c,i}} \Delta t_{ij}}{\Gamma\left(1 + \frac{1}{\beta_i}\right) \sum_{j=1}^{n_{c,i}} \frac{1}{n_{c,ij}}} \tag{2.11}$$

where $n_{c,ij}$ is given in Equation 2.10. Using Equation 2.9, the MTTF of the processing element is as follows:

$$\mu_i = \frac{\sum_{j=1}^{n_{c,i}} \Delta t_{ij}}{\sum_{j=1}^{n_{c,i}} \frac{1}{n_{c,ij}}}. \tag{2.12}$$

In conclusion, it is worth emphasizing that the reliability model requires detailed information about the thermal cycles to which the processing element is exposed, which is the topic of Chapter 3 and, in particular, Section 3.4 where dynamic steady-state temperature analysis is discussed in detail.

3 Analysis and Design with Certainty

In this chapter, we elaborate on deterministic system-level analysis and design of electronic systems. The techniques that we discuss here do not take uncertainty into account. However, these techniques provide an adequate foundation for those that do, which is the topic of the subsequent chapters.

3.1 Introduction

Although there are many aspects that are of significance to the design of an electronic system, our interest revolves primarily around those that are related to power and temperature for the following reason: power consumption and heat dissipation are of great importance. Power translates to energy, and energy translates to hours of battery life and to electricity bills. Temperature, on the other hand, is one of the major causes of permanent damage [37], which necessitates adequate cooling equipment and thereby escalates product expenses [14]. The situation is complicated even further by the interdependence between power and temperature, which is discussed in Section 2.2 and Section 3.5: higher power leads to higher temperature, and higher temperature to higher power [75]. Consequently, taking power and temperature into consideration is key to achieving effectiveness, efficiency, and robustness.

In the following, we consider temperature analysis; it is important to realize, however, that power analysis is inseparable from temperature analysis due to the aforementioned power-temperature interplay. Hence, power analysis is always implied whenever we discuss temperature analysis, and vice versa.

There are two types of temperature analysis: (a) transient analysis and (b) steady-state analysis. The first records the temperatures that the system under consideration traverses in an arbitrary time interval starting from an arbitrary initial condition when it is exposed to a spatially and temporally arbi-

trary power distribution. The second records the temperature that the system attains and retains once it has reached a thermal equilibrium under a power distribution that is spatially arbitrary but temporally constant or repeating.

Steady-state analysis can be further classified into two categories: (a) static analysis and (b) dynamic analysis. The first is concerned with temporally constant power distributions, that is, with those that do not change over time. In this case, the resulting temperature distributions do not change over time either. The second is concerned with temporally arbitrary power distributions that are periodic, that is, with those that repeat with a certain periodicity. In this case, the resulting temperature distributions change over time with the same periods as the corresponding repeating power distributions. The scenario being considered is that the system is exposed to a periodic workload or to a workload that can be approximated as periodic. Prominent examples that have such periodic behaviors are various multimedia applications.

3.2 Transient Analysis

The goal of transient analysis is to compute the temperature profile $\mathbf{Q}$, which is defined in Equation 2.2, that corresponds to a given power profile $\mathbf{P}$, which is defined in Equation 2.1, by solving the system given in Equation 2.5. Traditionally, this operation is undertaken numerically [100]; however, we are interested in obtaining and working with an analytical solution to the system, since such a solution has many advantages, as we shall see later on.

3.2.1 Previous Work

Let us discuss an analytical approach to solving the system of differential equations given in Equation 2.5a. To begin with, the system is rewritten as follows:

$$\frac{\mathrm{d}\tilde{\boldsymbol{s}}(t)}{\mathrm{d}t} = \tilde{\mathbf{A}}\tilde{\boldsymbol{s}}(t) + \mathbf{C}^{-1}\tilde{\mathbf{B}}\boldsymbol{p}(t)$$

where

$$\tilde{\mathbf{A}} = -\mathbf{C}^{-1}\mathbf{G}.$$

Next, we apply a technique that is taken from the family of exponential integrators, which have good stability properties; the interested reader is referred to [48] for an overview. Multiplying both sides of the above system of differential equations by $e^{-\tilde{\mathbf{A}}t}$ and noting that

$$e^{-\tilde{\mathbf{A}}t}\frac{\mathrm{d}\tilde{\boldsymbol{s}}(t)}{\mathrm{d}t} = \frac{\mathrm{d}e^{-\tilde{\mathbf{A}}t}\tilde{\boldsymbol{s}}(t)}{\mathrm{d}t} + e^{-\tilde{\mathbf{A}}t}\tilde{\mathbf{A}}\tilde{\boldsymbol{s}}(t),$$

we obtain

$$\tilde{\boldsymbol{s}}(t) = e^{\tilde{\mathbf{A}}t}\tilde{\boldsymbol{s}}(0) + e^{\tilde{\mathbf{A}}t}\int_0^t e^{-\tilde{\mathbf{A}}\tau}\mathbf{C}^{-1}\tilde{\mathbf{B}}\boldsymbol{p}(\tau)\mathrm{d}\tau,$$

which is the solution at time t starting from the initial condition $\tilde{\boldsymbol{s}}(0)$ at time 0.

Imagine now that the power consumption of the processing elements does not change over time: $\boldsymbol{p}(t) = \boldsymbol{p}$. In this case, the system is a system of linear differential equations that has the following analytical solution:

$$\tilde{\boldsymbol{s}}(t) = e^{\tilde{\mathbf{A}}t}\tilde{\boldsymbol{s}}(0) + \tilde{\mathbf{A}}^{-1}(e^{\tilde{\mathbf{A}}t} - \mathbf{I})\mathbf{C}^{-1}\tilde{\mathbf{B}}\boldsymbol{p} \tag{3.1}$$

where $\mathbf{I}$ is the identity matrix.

Suppose now that the sampling interval Δt of $\mathbf{P}$ is small enough so that the power consumption in each interval $[t_i, t_{i+1})$ can be reasonably approximated by a constant equal to $\boldsymbol{p}_i = \boldsymbol{p}(t_i)$. The corresponding $\mathbf{Q}$ can then be found by applying the following recurrence derived from Equation 3.1:

$$\tilde{\boldsymbol{s}}_i = \tilde{\mathbf{E}}\tilde{\boldsymbol{s}}_{i-1} + \tilde{\mathbf{F}}\boldsymbol{p}_i \tag{3.2}$$

for $i = 1, \ldots, n_s$ where

$$\begin{aligned}
&\tilde{\boldsymbol{s}}_0 = \mathbf{0} = (0, \ldots, 0),\\
&\tilde{\mathbf{E}} = e^{\tilde{\mathbf{A}}\Delta t}, \text{ and}\\
&\tilde{\mathbf{F}} = \tilde{\mathbf{A}}^{-1}(e^{\tilde{\mathbf{A}}\Delta t} - \mathbf{I})\mathbf{C}^{-1}\tilde{\mathbf{B}}.
\end{aligned}$$

Note that, in order to obtain the actual $\mathbf{Q}$, the recurrence should be followed by Equation 2.5b, which involves two trivial algebraic operations.

Similarly to the observations made in [85, 109], our experience shows that the approach to transient analysis described above provides a significant performance improvement compared to iterative solutions to systems of ordinary differential equations, such as the fourth-order Runge–Kutta method [87]. However, there is still room for improvement, as we discuss next.

3.2.2 Proposed Solution

Even though the matrices $\tilde{\mathbf{E}}$ and $\tilde{\mathbf{F}}$ have to be computed only once, they necessitate two computationally problematic operations: the matrix exponential and matrix inverse involving $\tilde{\mathbf{A}} \in \mathbb{R}^{n_n \times n_n}$, which is a generic matrix. It is preferable to have a symmetric matrix $\mathbf{A} \in \mathbb{R}^{n_n \times n_n}$ when these operations are concerned, since such a matrix admits the eigendecomposition, which can be seen in Equation A.1. Having computed such a decomposition, the calculation of the matrix exponential and matrix inverse becomes trivial as follows:

$$e^{\mathbf{A}\Delta t} = \mathbf{U}e^{\mathbf{\Lambda}\Delta t}\mathbf{U}^{\mathrm{T}} = \mathbf{U}\,\mathrm{diag}\left(e^{\lambda_1\Delta t}, \ldots, e^{\lambda_{n_n}\Delta t}\right)\mathbf{U}^{\mathrm{T}} \text{ and} \tag{3.3}$$

$$\mathbf{A}^{-1} = \mathbf{U}\mathbf{\Lambda}^{-1}\mathbf{U}^{\mathrm{T}} = \mathbf{U}\,\mathrm{diag}\left(\lambda_1^{-1}, \ldots, \lambda_{n_n}^{-1}\right)\mathbf{U}^{\mathrm{T}}. \tag{3.4}$$

In order to obtain such an $\mathbf{A}$, we propose to perform an auxiliary transformation. Recall first that the conductance matrix $\mathbf{G}$ is a symmetric matrix,

which, intuitively, is due to the fact that, if node i is connected to node j with a certain conductance, node j is also connected to node i with the same conductance; see Figure 2.1. However, $\tilde{\mathbf{A}} = -\mathbf{C}^{-1}\mathbf{G}$ does not have this property. The desired symmetry can be kept intact using the following substitution:

$$\begin{aligned}\boldsymbol{s}(t) &= \mathbf{C}^{\frac{1}{2}}\tilde{\boldsymbol{s}}(t) \text{ and}\\ \mathbf{A} &= -\mathbf{C}^{-\frac{1}{2}}\mathbf{G}\mathbf{C}^{-\frac{1}{2}}\end{aligned}$$

where $\mathbf{A}$ is symmetric, since

$$\mathbf{A}^{\mathrm{T}} = -(\mathbf{C}^{-\frac{1}{2}}\mathbf{G}\mathbf{C}^{-\frac{1}{2}})^{\mathrm{T}} = -(\mathbf{C}^{-\frac{1}{2}})^{\mathrm{T}}\mathbf{G}^{\mathrm{T}}(\mathbf{C}^{-\frac{1}{2}})^{\mathrm{T}} = -\mathbf{C}^{-\frac{1}{2}}\mathbf{G}\mathbf{C}^{-\frac{1}{2}} = \mathbf{A}.$$

Consequently, Equation 2.5 is rewritten as follows:

$$\begin{cases}\dfrac{\mathrm{d}\boldsymbol{s}(t)}{\mathrm{d}t} = \mathbf{A}\boldsymbol{s}(t) + \mathbf{B}\boldsymbol{p}(t) & \text{(3.5a)}\\ \boldsymbol{q}(t) = \mathbf{B}^{\mathrm{T}}\boldsymbol{s}(t) + \boldsymbol{q}_{\mathrm{amb}} & \text{(3.5b)}\end{cases}$$

where

$$\mathbf{B} = \mathbf{C}^{-\frac{1}{2}}\tilde{\mathbf{B}}.$$

Similarly, the solution in Equation 3.1 becomes

$$\boldsymbol{s}(t) = e^{\mathbf{A}t}\boldsymbol{s}_0 + \mathbf{A}^{-1}(e^{\mathbf{A}t} - \mathbf{I})\mathbf{B}\boldsymbol{p},$$

and the recurrence in Equation 3.2 becomes

$$\boldsymbol{s}_i = \mathbf{E}\boldsymbol{s}_{i-1} + \mathbf{F}\boldsymbol{p}_i \tag{3.6}$$

for $i = 1, \ldots, n_s$ where

$$\begin{aligned}&\boldsymbol{s}_0 = \mathbf{0},\\ &\mathbf{E} = e^{\mathbf{A}\Delta t}, \text{ and}\\ &\mathbf{F} = \mathbf{A}^{-1}\left(e^{\mathbf{A}\Delta t} - \mathbf{I}\right)\mathbf{B}.\end{aligned}$$

Using the eigendecomposition in Equation A.1, the last equation can be efficiently computed in the following way:

$$\mathbf{F} = \mathbf{U}\,\mathrm{diag}\left(\frac{e^{\lambda_1\Delta t} - 1}{\lambda_1}, \ldots, \frac{e^{\lambda_{n_n}\Delta t} - 1}{\lambda_{n_n}}\right)\mathbf{U}^{\mathrm{T}}\mathbf{B}.$$

As before, the recurrence in Equation 3.6 should be followed by Equation 3.5b in order to obtain $\mathbf{Q}$. The above auxiliary transformation is helpful not only for transient analysis but also in other contexts, as we shall see later on.

Let us note at this point that there have been attempts to simplify the temperature model by making additional assumptions in order to reduce the size

of the circuit, thereby speeding up the solution process. For instance, the techniques proposed in [3, 89] are targeting single-core platforms, and the approach described in [91] is aimed at homogeneous platforms and applications where the execution times of tasks are comparable with the thermal time constant of the thermal package, which is in the order of 100 s. Such techniques can be combined with what we present in this thesis as long as $\mathbf{C}$ remains a diagonal matrix, and $\mathbf{G}$ remains a symmetric positive definite matrix.

3.3 Static Steady-State Analysis

The goal of static steady-state analysis is to calculate the temperature of the thermal equilibrium that the system reaches when its power consumption stays at a certain constant level $\boldsymbol{p}$ for a sufficiently long period of time. In this case, there are no dynamics, which means that the derivative in Equation 3.5a is zero. The static steady-state temperature $\boldsymbol{q}$ can then be calculated as

$$\boldsymbol{s} = -\mathbf{A}^{-1}\mathbf{B}\boldsymbol{p}$$

followed by Equation 3.5b where $\mathbf{A}^{-1}$ is computed using Equation 3.4. In this case, $\boldsymbol{p}$ and $\boldsymbol{q}$ can be viewed as a degenerate power profile $\mathbf{P}$ and a degenerate temperature profile $\mathbf{Q}$, respectively, in the sense that $n_s = 1$.

The static steady-state temperature produced by static steady-state analysis is frequently used as a computationally cheap approximation of the thermal behavior of the system. For instance, given a power profile $\mathbf{P}$ with $n_s > 1$, one might simply compute the average power vector $\boldsymbol{p}$, perform static steady-state analysis, and utilize the resulting static $\boldsymbol{q}$ as a guide inside a temperature-aware design-space-exploration procedure. However, this approximation is of limited applicability. It assumes that the system under consideration functions at one constant temperature at each spatial location, which does not hold in the majority of cases, since, in reality, power consumption readily changes.

3.4 Dynamic Steady-State Analysis

Dynamic steady-state analysis addresses the shortcomings of static steady-state analysis in one particular but important context. It tackles the scenario where power consumption follows a periodic pattern. In this case, after a sufficiently long time, the system does not reach a static steady state but instead a dynamic steady state where temperature starts to exhibit a periodic pattern following the periodic pattern of power. The goal of the analysis is then to find the periodic temperature profile $\mathbf{Q}$, called the dynamic steady-state temperature profile, that corresponds to a given periodic power profile $\mathbf{P}$.

In the case of applications that exhibit periodic or close to periodic behaviors, this analysis is of particular importance. Any design optimization has to be performed such that the efficiency and reliability of the system at hand are

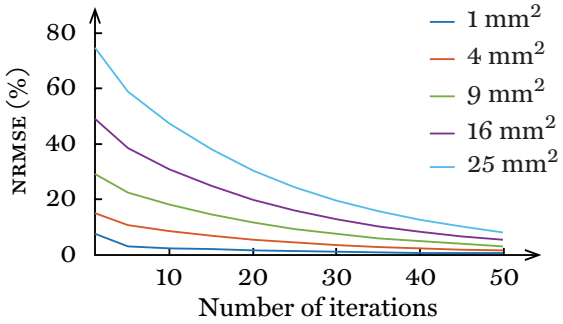

Figure 3.1: Accuracy of the iterative transient approximation to dynamic steady-state temperature analysis

maximized by considering not a relatively short transient time interval at the system's start but the context in which the system is to function over a long period of time, which is exactly the dynamic steady state of the system.

A prominent example of a design task for which the analysis is of central importance is reliability optimization targeted at mitigating thermal-cycling fatigue [37]. As noted in Section 2.4.2, in this context, the lifetime of the system is impacted not only by the average and maximum temperature but also by the amplitude and frequency of temperature oscillations. Thus, efficient reliability optimization depends on the availability of the actual dynamic steady-state temperature profile, which we discuss further in Section 3.6.

3.4.1 Previous Work

Let us elaborate on the two techniques that have been presented in the literature in order to calculate the dynamic steady-state temperature profile.

The first technique is referred to as the iterative transient approximation. In this case, an estimate of the profile is calculated by running a temperature simulator over a number of successive periods of the application until a sufficiently accurate approximation of the dynamic steady state is assumed to be attained [102]. The typical simulator of choice in this context is HotSpot [100], which is the state of the art in system-level temperature analysis of electronic systems; see, for instance, [24, 50, 73, 75, 83, 86, 102, 109, 118]. The simulator performs transient temperature analysis by solving the system in Equation 2.5 numerically by means of the fourth-order Runge–Kutta method [87].

The number of iterations required to reach the dynamic steady state depends on the thermal characteristics of the platform. In order to illustrate this aspect, consider an application with a period of 500 ms running on five hypothetical platforms with processing-element areas of 1–25 mm^2. Let us simulate 50 successive periods of the application via HotSpot with its default settings

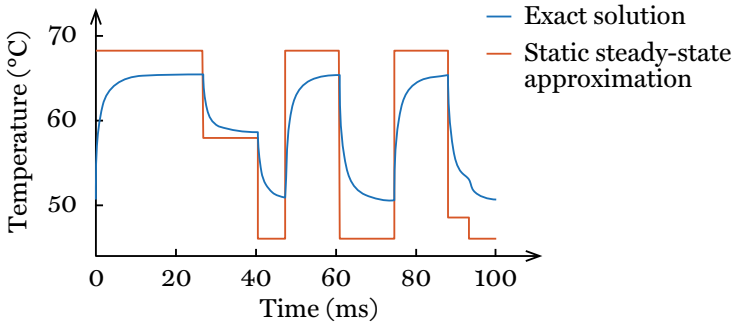

Figure 3.2: Example of the static steady-state approximation to dynamic steady-state temperature analysis

and compare the resulting approximations in each period with the actual dynamic steady-state temperature profiles—whose calculation will be discussed in Section 3.4.2—using the normalized root-mean-square error (NRMSE). The comparison is shown in Figure 3.1. It can be observed that the number of successive periods over which transient analysis has to be performed in order to achieve a satisfactory level of accuracy is significant for the majority of the configurations, which entails large computation times. For instance, for an area of 9 mm^2, even after 15 iterations, the NRMSE is still close to 20%. Using the analytical approach to transient analysis presented in Section 3.2, the iterative transient approximation can be sped up; however, the large number of iterations still makes the computational cost unreasonably high, as discussed further in Section 3.4.3. Moreover, the approach provides no guarantees on the resulting accuracy, since there is no reliable metric for measuring the proximity to the actual dynamic steady-state temperature profile.

The second technique is referred to as the static steady-state approximation. It is a crude but fast technique proposed in [50]. The approach forgoes transient analysis and resorts to static steady-state analysis instead. Specifically, it is assumed that, in each time interval where the system's power consumption is constant, the system instantaneously reaches a static steady state. The result of this procedure is a stepwise temperature curve where each step corresponds to the static steady-state temperature that would be reached if the corresponding power was applied for a sufficiently long period of time.

An example of such an approximation, along with the actual dynamic steady-state temperature profile, for an application with 10 tasks and a period of 100 ms is given in Figure 3.2. The processing-element area is 25 mm^2 in this case. The reduced accuracy of this technique is due to the mismatch between the actual temperature within each interval and the static steady-state temperature. The inaccuracy depends on the thermal characteristics of the platform and on the application itself. In order to illustrate this, let us simulate five ap-

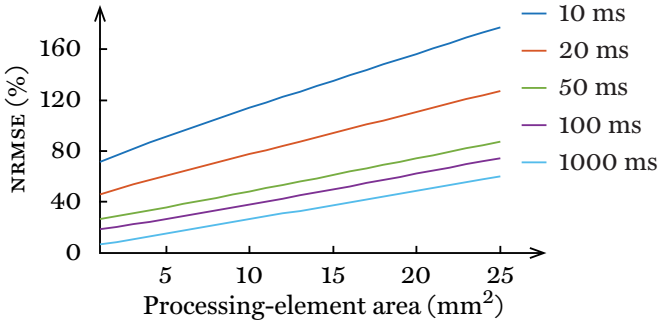

Figure 3.3: Accuracy of the static steady-state approximation to dynamic steady-state temperature analysis

plications with periods of 10–1000 ms running on five platforms with an area per processing element of 1–25 mm^2. The errors are shown in Figure 3.3. It can be seen that, for example, for an area of 10 mm^2 and a period of 100 ms, the NRMSE of this approximation technique is close to 40%.

To conclude, the current state-of-the-art techniques for dynamic steady-state temperature analysis are slow and inaccurate. This state of affairs makes them difficult and dangerous from the standpoint of design optimization.

3.4.2 Proposed Solution

In this subsection, we formalize the problem of dynamic steady-state temperature analysis and develop an exact and, moreover, computationally efficient solution to this problem, which eliminates the shortcomings of the state-of-the-art solutions discussed in the previous subsection.

Consider the temperature model in Equation 3.5 and the corresponding recurrence in Equation 3.6. The key condition of a dynamic steady state is

$$\boldsymbol{s}_0 = \boldsymbol{s}_{n_s}. \tag{3.7}$$

The above condition means that, once the dynamic steady state has been reached, the electronic system starts returning to its initial state at the end of each iteration, which is what makes the system's behavior periodic. Then, using Equation 3.6, the dynamic steady-state temperature profile can be computed by solving the following system of linear equations:

$$\begin{cases} \boldsymbol{s}_1 - \mathbf{E}\boldsymbol{s}_{n_s} & = \mathbf{F}\boldsymbol{p}_1 \\ \boldsymbol{s}_2 - \mathbf{E}\boldsymbol{s}_1 & = \mathbf{F}\boldsymbol{p}_2 \\ \dots & \\ \boldsymbol{s}_{n_s} - \mathbf{E}\boldsymbol{s}_{n_s-1} & = \mathbf{F}\boldsymbol{p}_{n_s} \end{cases}$$

where the first equation enforces the boundary condition in Equation 3.7. In order to get the big picture, the system can be rewritten as follows:

$$\underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & -\mathbf{E} \\ -\mathbf{E} & \mathbf{I} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -\mathbf{E} & \mathbf{I} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & -\mathbf{E} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & -\mathbf{E} & \mathbf{I} \end{bmatrix}}_{\mathbb{A}} \underbrace{\begin{bmatrix} \boldsymbol{s}_1 \\ \boldsymbol{s}_2 \\ \boldsymbol{s}_3 \\ \cdots \\ \boldsymbol{s}_{n_s-2} \\ \boldsymbol{s}_{n_s-1} \\ \boldsymbol{s}_{n_s} \end{bmatrix}}_{\mathbb{X}} = \underbrace{\begin{bmatrix} \mathbf{F}\boldsymbol{p}_1 \\ \mathbf{F}\boldsymbol{p}_2 \\ \mathbf{F}\boldsymbol{p}_3 \\ \cdots \\ \mathbf{F}\boldsymbol{p}_{n_s-2} \\ \mathbf{F}\boldsymbol{p}_{n_s-1} \\ \mathbf{F}\boldsymbol{p}_{n_s} \end{bmatrix}}_{\mathbb{B}} \tag{3.8}$$

where $\mathbb{A}$ is an $n_n n_s \times n_n n_s$ matrix, and $\mathbb{X}$ and $\mathbb{B}$ are $n_n n_s$-element vectors.

The most direct way to solve the system in Equation 3.8 is to use a dense solver, such as the LU decomposition [87]. However, since $\mathbb{A}$ is a sparse matrix, a more appropriate approach is to employ a sparse solver, such as the unsymmetric multifrontal method [29]. The computational complexity of such solutions is $\mathcal{O}(n_s^3 n_n^3)$ [87] where n_n is the number of nodes in the thermal RC circuit, and n_s is the number of samples in the power profile. The problem, however, is that the system to solve can be extremely large, which is due primarily to n_s. In such cases, direct solvers are prohibitively slow and require an enormous amount of memory. Therefore, we do not discuss them any further.

Another potential approach is leveraging iterative methods for solving systems of linear equations, such as the Jacobi or Gauss–Seidel method [87]. Such methods are designed to overcome the problems of direct solvers, and they are consequently applicable for very large systems. However, the most important issue with these methods is their convergence. In our experiments, we have not observed any advantages of using these methods compared to the other techniques. Thus, they are excluded from further discussion.

Yet another solution can be obtained by observing that $\mathbb{A}$ in Equation 3.8 is, in fact, a block Toeplitz matrix and, moreover, a block-circulant matrix, in which each block row is rotated one block element to the right relative to the preceding block row. This observation leads to a range of possible techniques for solving the system of equations shown in Equation 3.8, such as the fast Fourier transform [30] used in the experiments reported in Section 3.4.3.

The major problem with the aforementioned techniques is that (a) the sparseness of $\mathbb{A}$ is not taken into account, or (b) the specific structure of $\mathbb{A}$ is ignored, which results in inefficient and, in some cases, inaccurate computations. Let us now develop a solution that does not have these issues.

Have a careful look at the structure of $\mathbb{A}$ in Equation 3.8. The nonzero elements are located only on the block diagonal, the subdiagonal attached to the block diagonal, and the superdiagonal in the top-right corner of the matrix. Linear systems with similar structures arise in boundary value problems of ordinary differential equations, and the technique to solve them is to form a so-called condensed equation or condensed system [104], as we describe next.

To begin with, let

$$\boldsymbol{v}_i = \mathbf{F}\boldsymbol{p}_i$$

for $i = 1, \ldots, n_s$. Equation 3.6 can then be rewritten as follows:

$$\boldsymbol{s}_i = \mathbf{E}\boldsymbol{s}_{i-1} + \boldsymbol{v}_i \tag{3.9}$$

for $i = 1, \ldots, n_s$. Applying this formula recursively starting from $\boldsymbol{s}_0$ leads to

$$\boldsymbol{s}_i = \mathbf{E}^i\boldsymbol{s}_0 + \boldsymbol{w}_i$$

for $i = 1, \ldots, n_s$. In the above, $\boldsymbol{w}_i$ is an auxiliary recurrence defined by

$$\boldsymbol{w}_i = \mathbf{E}\boldsymbol{w}_{i-1} + \boldsymbol{v}_i \tag{3.10}$$

for $i = 1, \ldots, n_s$ where

$$\boldsymbol{w}_0 = \mathbf{0}.$$

After taking n_s steps, we arrive at the following state vector:

$$\boldsymbol{s}_{n_s} = \mathbf{E}^{n_s}\boldsymbol{s}_0 + \boldsymbol{w}_{n_s}.$$

Taking into account the boundary condition given in Equation 3.7, we obtain the following system of linear equations:

$$(\mathbf{I} - \mathbf{E}^{n_s})\boldsymbol{s}_{n_s} = \boldsymbol{w}_{n_s}.$$

Since $\mathbf{E}$ is the matrix exponential, which can be seen in Equation 3.3, the above system can be rewritten as follows:

$$(\mathbf{I} - \mathbf{U}e^{\boldsymbol{\Lambda}\tau}\mathbf{U}^{\mathrm{T}})\boldsymbol{s}_{n_s} = \boldsymbol{w}_{n_s}$$

where $\tau = n_s\Delta t$ is the period of the power profile $\mathbf{P}$. By splitting the identity matrix $\mathbf{I}$ into $\mathbf{U}\mathbf{U}^{\mathrm{T}}$, we obtain the following solution to the system:

$$\begin{aligned}\boldsymbol{s}_{n_s} &= \mathbf{U}(\mathbf{I} - e^{\boldsymbol{\Lambda}\tau})^{-1}\mathbf{U}^{\mathrm{T}}\boldsymbol{w}_{n_s} \\ &= \mathbf{U}\,\mathrm{diag}\left(\frac{1}{1-e^{\lambda_1\tau}}, \ldots, \frac{1}{1-e^{\lambda_{n_s}\tau}}\right)\mathbf{U}^{\mathrm{T}}\boldsymbol{w}_{n_s}.\end{aligned}$$

The above equation yields not only the final state vector $\boldsymbol{s}_{n_s}$ but also the initial one $\boldsymbol{s}_0$. Consequently, the rest of the state vectors $\{\boldsymbol{s}_i\}_{i=1}^{n_s-1}$ can be successively found by means of Equation 3.9 where each $\boldsymbol{v}_i$ has already been calculated when computing $\boldsymbol{w}_{n_s}$. The last step of the solution is to compute the actual dynamic steady-state temperature profile $\mathbf{Q}$ by applying Equation 3.5b.

It can be seen that the solution to the $n_n n_s \times n_n n_s$ system in Equation 3.8 has been reduced to the two trivial recurrences in Equation 3.9 and Equation 3.10 that traverse the n_s steps of the power profile $\mathbf{P}$. The pseudocode for this solution is given in Algorithm 3.1, and its key aspects are as follows.

Algorithm 3.1: Calculation of a dynamic steady-state temperature profile

Input: $\mathbf{P} \in \mathbb{R}^{n_p \times n_s}$
Output: $\mathbf{Q} \in \mathbb{R}^{n_p \times n_s}$
1: $\mathbf{V} \leftarrow \mathbf{FP}$ // an auxiliary variable
2: $\boldsymbol{w} \leftarrow \mathbf{V}(:, 1)$ // an auxiliary variable
3: **for** $i \leftarrow 2$ **to** n_s **do**
4: $\quad \boldsymbol{w} \leftarrow \mathbf{E}\boldsymbol{w} + \mathbf{V}(:, i)$
5: **end for**
6: $\mathbf{S}(:, n_s) \leftarrow \mathbf{U}\,\mathrm{diag}(\frac{1}{1-e^{\lambda_i \tau}})\mathbf{U}^{\mathrm{T}}\boldsymbol{w}$ // a matrix version of $\boldsymbol{s}$
7: **for** $i \leftarrow 1$ **to** $n_s - 1$ **do**
8: $\quad \mathbf{S}(:, i) \leftarrow \mathbf{E}\mathbf{S}(:, i-1) + \mathbf{V}(:, i)$
9: **end for**
10: $\mathbf{Q} \leftarrow \mathbf{B}^{\mathrm{T}}\mathbf{S} + \mathbf{Q}_{\mathrm{amb}}$ // a matrix version of $\boldsymbol{q}_{\mathrm{amb}}$
11: **return** $\mathbf{Q}$

Line 2–5: Equation 3.10 is recursively evaluated for all time steps of the input power profile $\mathbf{P}$ in order to calculate the auxiliary vector $\boldsymbol{w}_{n_s}$.

Line 6: The calculated $\boldsymbol{w}_{n_s}$ is utilized in order to compute the final state vector $\boldsymbol{s}_{n_s}$, which is then stored in the last column of an $n_n \times n_s$ matrix $\mathbf{S}$.

Line 7–9: Using $\boldsymbol{s}_{n_s}$ or, equivalently, $\boldsymbol{s}_0$, Equation 3.9 is recursively evaluated in order to calculate the remaining state vectors, that is, $\{\boldsymbol{s}_i\}_{i=1}^{n_s-1}$. Note that, in the first iteration, $\mathbf{S}(:, 0)$ should be understood as $\mathbf{S}(:, n_s)$.

Line 10: Equation 3.5b is applied in order to compute the desired $\mathbf{Q}$.

It is worth noting that the auxiliary transformation presented in Section 3.2.2 and the accompanying eigendecomposition in Equation A.1 have substantially simplified the calculations associated with dynamic steady-state analysis. Note also that the eigendecomposition along with $\mathbf{E}$ and $\mathbf{F}$ are computed only once for a particular thermal RC circuit and can be considered to be given together with the circuit. In other words, these quantities stay the same when different power profiles are to be analyzed, which is particularly advantageous when an intensive design-space-exploration procedure is concerned.

The computational complexity of the whole procedure is estimated to be

$$\mathcal{O}\left(n_s n_n^2 + n_n^3\right).$$

The complexity is linear with respect to n_s, which is important, since n_s is typically much larger than n_n, that is, the number of thermal nodes.

3.4.3 Experimental Results

Let us assess the performance of the solution to dynamic steady-state analysis proposed in Section 3.4.2. All the experiments are conducted on a GNU/Linux machine equipped with an Intel Core i7 3.4 GHz and 8 GB of RAM.

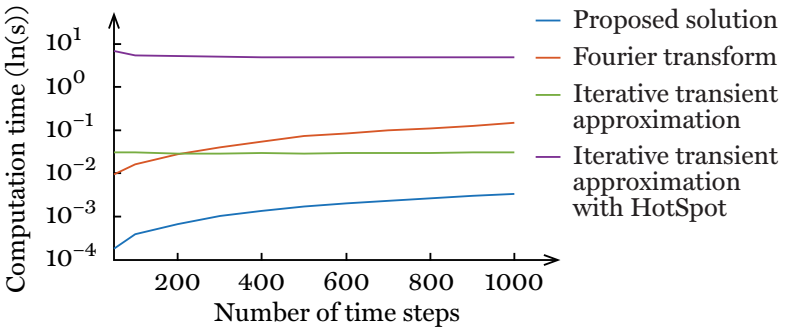

Figure 3.4: Computational speed of different solutions to dynamic steady-state analysis with respect to the number of time steps

Thermal RC circuits are constructed by means of HotSpot [100] with its default configuration, and they follow the principle described in Section 2.3. The sampling interval Δt of power and temperature profiles is set to 1 ms.

For purposes of comparison, we consider two alternative techniques. Namely, we use the one based on iterative transient analysis, which is introduced in Section 3.4.1, and the one based on the fast Fourier transform, which is mentioned in Section 3.4.2, since they are comparable with the proposed technique in terms of accuracy. In the case of transient analysis, we evaluate both the fast analytical solution described in Section 3.2.2 and the one implemented in HotSpot, and the corresponding iterative calculation is performed until the NRMSE relative to the dynamic steady-state temperature profile computed by means of the proposed method, which is exact, is less than 1%.

First of all, we vary the period τ of power and temperature profiles—and, therefore, the number of samples n_s that they contain—while keeping the architecture fixed, which is a quad-core platform with a processing-element area of 4 mm^2. The results of this experiment are depicted in Figure 3.4 on a semilogarithmic scale. It can be seen that the proposed technique is 9–170 times faster than iterative transient analysis with the analytical solution and roughly 5000 times faster than iterative transient analysis with HotSpot.

In the second experiment, we evaluate the scaling properties of our method with respect to the number of processing elements n_p. The period is fixed to 500 ms, which results in 500 time steps. The results are shown in Figure 3.5. It can be observed that the proposed technique provides a significant performance improvement relative to the alternative solutions.

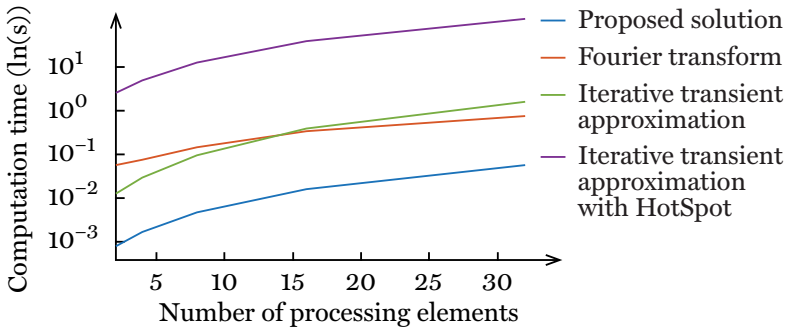

Figure 3.5: Computational speed of different solutions to dynamic steady-state analysis with respect to the number of processing elements

3.5 Power-Temperature Interdependence

So far, the interdependence between power and temperature, which is introduced in Section 2.2, has been ignored. In order to properly take it into account, several approaches can be employed as follows.

The first approach is to calculate the power and temperature profiles several times in turns until the latter converges. In this case, we obtain a series of pairs of a power profile and a temperature profile

$$\{(\mathbf{P}_i, \mathbf{Q}_i) : i = 1, 2, \dots\}.$$

For each new temperature profile $\mathbf{Q}_i$—which is computed by performing transient, static steady-state, or dynamic steady-state analysis as usual—a new power profile $\mathbf{P}_i$ is obtained by recalculating the static power component and adding it to the dynamic one, which we write as follows:

$$\mathbf{P}_i = \mathbf{P}_{\text{dyn}} + \mathbf{P}_{\text{stat},i}(\mathbf{Q}_{i-1})$$

where $\mathbf{P}_{\text{stat},i}$ is computed using $\mathbf{Q}_{i-1}$. The process continues until a stopping condition is satisfied, such as when the difference between two successive temperature profiles $\mathbf{Q}_{i-1}$ and $\mathbf{Q}_i$ drops below a predefined threshold.

In the case of dynamic steady-state analysis, which is discussed in Section 3.4, the pseudocode for the aforementioned iterative procedure is listed in Algorithm 3.2. The two main steps of this algorithm are as follows.

Line 3: The power profile $\mathbf{P}$ is updated using the current temperature profile $\mathbf{Q}$. In the first iteration, the ambient temperature is used for this purpose.

Line 4: The temperature profile $\mathbf{Q}$ is updated using the current power profile $\mathbf{P}$ by means of Algorithm 3.1, which is discussed in Section 3.4.2.

In order to give a better sense of the convergence rate of the iterative procedure, let us give an example. According to our experience, it typically takes 4–7 iterations until the uniform norm of the difference between two successive

Algorithm 3.2: Calculation of dynamic steady-state power and temperature profiles considering the power-temperature interdependence

Input: $\mathbf{P}_{\text{dyn}} \in \mathbb{R}^{n_p \times n_s}$
Output: $\mathbf{P} \in \mathbb{R}^{n_p \times n_s}$, $\mathbf{Q} \in \mathbb{R}^{n_p \times n_s}$
1: $\mathbf{Q} \leftarrow \mathbf{Q}_{\text{amb}}$ // a matrix version of $\boldsymbol{q}_{\text{amb}}$
2: **repeat**
3: $\quad \mathbf{P} \leftarrow \mathbf{P}_{\text{dyn}} + \mathbf{P}_{\text{stat}}(\mathbf{Q})$
4: $\quad \mathbf{Q} \leftarrow$ Algorithm 3.1($\mathbf{P}$)
5: **until** a stopping condition is satisfied
6: **return** $\mathbf{P}$, $\mathbf{Q}$

dynamic steady-state temperature profiles becomes smaller than 0.5°C, which is considered to be sufficiently accurate for many applications.

In the case of transient analysis, the iterative procedure can also be done on a step-by-step basis, even with only one iteration. Specifically, at each step of the iterative process in Equation 3.6, one can simply calculate the static power component at the current temperature, add it to the dynamic one, and use the result for evaluating the next temperature. Hence, the recurrence becomes

$$\boldsymbol{s}_i = \mathbf{E}\boldsymbol{s}_{i-1} + \mathbf{F}(\boldsymbol{p}_{\text{dyn},i} + \boldsymbol{p}_{\text{stat},i}(\boldsymbol{q}_{i-1}))$$

for $i = 1, \ldots, n_s$ where, as indicated, $\boldsymbol{p}_{\text{stat},i}$ is computed using $\boldsymbol{q}_{i-1}$.

The second approach is to use a linear approximation of static power, which can be formalized as follows:

$$\boldsymbol{p}_{\text{stat}}(t) = \hat{\mathbf{A}}\boldsymbol{q}(t) + \hat{\boldsymbol{b}}$$

where $\hat{\mathbf{A}} \in \mathbb{R}^{n_n \times n_n}$ and $\hat{\boldsymbol{b}} \in \mathbb{R}^{n_n}$ are a diagonal matrix and a vector to be fitted. It can be shown that the linear approximation keeps the original system of differential equations in Equation 2.5a almost intact. More precisely, it becomes

$$\mathbf{C}\frac{\mathrm{d}\tilde{\boldsymbol{s}}(t)}{\mathrm{d}t} + \hat{\mathbf{G}}\tilde{\boldsymbol{s}}(t) = \tilde{\mathbf{B}}\hat{\boldsymbol{p}}(t)$$

where

$$\hat{\mathbf{G}} = \mathbf{G} - \tilde{\mathbf{B}}\hat{\mathbf{A}}\tilde{\mathbf{B}}^{\mathrm{T}} \text{ and}$$
$$\hat{\boldsymbol{p}}(t) = \boldsymbol{p}_{\text{dyn}}(t) + \hat{\mathbf{A}}\boldsymbol{q}_{\text{amb}} + \hat{\boldsymbol{b}},$$

and $\hat{\mathbf{G}}$ has the same properties as $\mathbf{G}$. Therefore, all the solutions that have been derived in this chapter remain perfectly valid. Moreover, as shown in [75], despite its simplicity, the approximation is relatively accurate and, if needed, can be improved by considering multiple linear segments.

In order to evaluate the linearization, we consider a number of hypothetical platforms with 2–32 processing elements and undertake dynamic steady-state

analysis. The experiment is performed with both the linear approximation and the full exponential model in Equation 2.4. For the former, the model is fitted using least squares [87] targeted at the range 40°C–80°C. For the latter, we use the iterative approach described earlier. The results show that the NRMSE is bounded by 2%, indicating that the linearization is adequately accurate.

It is important to note that, regardless of the approach utilized, power and temperature are analyzed simultaneously, as they are interdependent. One obtains not only a temperature profile but also the corresponding static or total power profile, and all profiles take account of the power-temperature interplay.

Lastly, let us mention that static power is not considered in the experiments given in Section 3.4.3. However, if it was taken into account using the linearization, the computation times would remain unchanged, and, if the iterative model was utilized, the computation times would increase proportionally for all the techniques, which would not affect any of the conclusions.

3.6 Reliability Optimization

In this section, we illustrate the importance of temperature analysis for design-space exploration and, more specifically, for reliability optimization.

Among the commonly studied failure mechanisms (see Section 1.1.3), thermal cycling arguably has the most prominent dependence on temperature: as noted earlier, not only the average and maximum temperature but also the amplitude and frequency of temperature oscillations have a huge impact on the lifetime of the system. With this in mind, we focus our attention on mitigating the damage caused by thermal cycling. To this end, relying on our solution to dynamic steady-state analysis presented in Section 3.4.2, we develop a thermal-cycling-aware technique for scheduling periodic applications. Note that, by exploring the design space in order to find configurations that reduce wear on the system, we implicitly address aging uncertainty, which is introduced in Section 1.1.3. However, this approach to aging uncertainty is suboptimal, which is discussed further and addressed in Section 5.11.

3.6.1 Motivational Example

Consider a periodic application with six tasks denoted by T1–T6 and a heterogeneous platform with two processing elements denoted by PE1 and PE2. The application's task graph is given in Figure 3.6 along with the execution times of the tasks for both PE1 and PE2. The period of the application is 60 ms.

The first alternative schedule and the resulting dynamic steady-state temperature profile are shown on the left-hand side of Figure 3.7 where the height of a task represents its power consumption. Note that the mapping of the tasks onto the processing elements is treated as a part of the schedule. The blue curve illustrates that PE1 is experiencing three thermal cycles; that is, there are three intervals where temperature starts from a certain value, reaches an

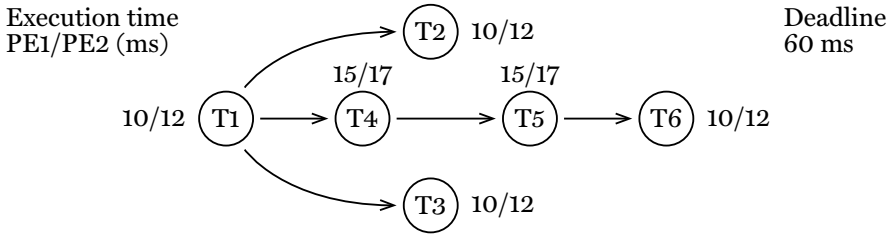

Figure 3.6: Task graph of an application along with the execution times of the tasks with respect to two processing elements

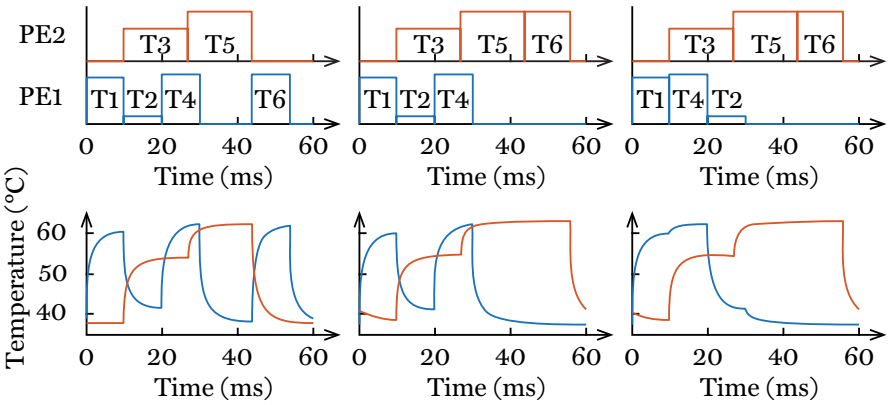

Figure 3.7: Alternative schedules, including mappings, of the application and the corresponding dynamic steady-state temperature profiles

extremum, and then comes back. If we move T6 to PE2, the number of cycles decreases to two, which can be seen in the middle of Figure 3.7. If we also swap T2 and T4, the number of cycles that PE1 undergoes drops to one, which is depicted on the right-hand side of Figure 3.7. According to the reliability model described in Section 2.4.2, these two changes improve the lifetime of the electronic system by around 45% and 55%, respectively, relative to the initial schedule, which can be seen on the left-hand side of Figure 3.7.

The above motivational example shows that, when exploring the design space, it is important to take into consideration the number of temperature fluctuations as well as their characteristics. In order to acquire this information, dynamic steady-state temperature analysis is required.

3.6.2 Problem Formulation

In addition to the description given in Section 2.1, the platform under consideration is supposed to execute a periodic application with n_t tasks. The applica-

tion is given as a directed acyclic graph whose vertices and edges correspond to the tasks and to data dependencies between these tasks, respectively; see Figure 3.6. Each pair of a task and a processing element is characterized by an execution time and a power consumption value, which are the characteristics that the task exhibits when it is assigned to the processing element.

The optimization objective is to find a schedule that maximizes the lifetime of the system under certain constraints. Formally, the objective is

$$\max_{\mathcal{S}} \min_{i=1}^{n_p} \mu_i(\mathcal{S}) \tag{3.11}$$

such that

$$\begin{aligned} &\tau(\mathcal{S}) \leq \tau_{\max} \text{ and} \\ &Q(\mathcal{S}) \leq q_{\max}. \end{aligned} \tag{3.12}$$

In the above formulae, $\mathcal{S}$ denotes a schedule, which specifies the starting times of the tasks as well as their mapping onto the processing elements; μ_i is the mean time to failure (MTTF) of processing element i, which is computed according to the reliability model presented in Section 2.4.2 and can be seen in Equation 2.12; τ stands for the end-to-end delay of the application; and

$$Q(\mathcal{S}) = \|\mathbf{Q}(\mathcal{S})\|_\infty$$

where $\mathbf{Q}$ is the dynamic steady-state temperature profile of the platform, and $\|\cdot\|_\infty$ denotes the uniform norm. The first constraint in Equation 3.12 enforces a deadline $\tau_{\max}$ on the schedule while the second constraint enforces a maximum temperature $q_{\max}$ on the resulting temperature profile $\mathbf{Q}$.

3.6.3 Proposed Solution

The application is scheduled by means of a static cyclic scheduler that follows the list scheduling policy [1]. The input to the scheduler is a vector mapping the tasks onto the processing elements and a vector assigning priorities to the tasks. The scheduler produces a vector prescribing starting times for the tasks, and this vector together with the given mapping constitute a schedule $\mathcal{S}$.

The optimization procedure is undertaken via a genetic algorithm [97]. In this biologically inspired paradigm, a population of chromosomes, which represent candidate solutions, is evolved through a number of generations in order to produce a chromosome with the best possible fitness, that is, a solution that maximizes the objective function. In our case, the fitness of a chromosome is calculated in accordance with Equation 3.11 and, more specifically, is set to

$$\min_{i=1}^{n_p} \mu_i(\mathcal{S}) \tag{3.13}$$

unless any of the constraints is violated, which we explain in the following.

Each chromosome contains $2n_t$ genes—twice the number of tasks—and it can be viewed as two concatenated vectors with n_t elements each. The first one encodes a mapping of the tasks onto the processing elements, and the other one a set of priorities for the tasks. The usage of this information will be discussed shortly. The population contains $4n_t$ chromosomes. In each generation, a number of chromosomes are chosen for breeding by a tournament selection with the number of competitors proportional to the population size. The chosen chromosomes undergo a two-point crossover with probability 0.8 and a uniform mutation with probability 0.01. The evolution mechanism follows the elitism model where the best chromosome always survives. The stopping condition is the absence of improvement within 200 successive generations.

The fitness of a chromosome is evaluated in a number of steps. First, the chromosome is decoded, and the mapping of the tasks onto the processing elements and the priorities of the tasks are fed to the scheduler. The scheduler produces a schedule $\mathcal{S}$. If the schedule violates the deadline constraint given in Equation 3.12, the chromosome is penalized proportionally to the amount of violation and is not processed further. Otherwise, based on the parameters of the processing elements and tasks, a power profile $\mathbf{P}$ is constructed, and the corresponding temperature profile $\mathbf{Q}$ is computed using our technique presented in Section 3.4.2. If the temperature profile violates the temperature constraint given in Equation 3.12, the chromosome is penalized proportionally to the amount of violation and is not processed further. Otherwise, the MTTF of each processing element is estimated according to Equation 2.12, and the fitness of the chromosome is set as shown in Equation 3.13.

3.6.4 Experimental Results

In this subsection, we evaluate our reliability optimization. We first consider a number of synthetic applications and then study a real-life one. The general experimental setup is the same as the one described in Section 3.4.3. All the configuration files used in the experiments are available online at [16].

Heterogeneous platforms and periodic applications are randomly generated by virtue of TGFF [34] in such a way that the execution times of tasks are uniformly distributed between 1 and 10 ms, and that the static power of processing elements accounts for around 40% of their total power [75]. The area of a processing element is set to 4 mm^2. The modeling of static power is based on the linearization discussed in Section 3.5. The maximum temperature $q_{\max}$ in Equation 3.12 is set to 100°C. In Equation 2.10, the Coffin–Manson exponent b_i is set to 6, the activation energy c_i to 0.5, and the elastic region $\Delta q_{0,ij}$ to 0 [37]; the value of the coefficient a_i in Equation 2.10 is irrelevant, since we are concerned with relative improvements, which will be explicated shortly.

The initial population of chromosomes is created partially randomly and partially based on a temperature-aware heuristic proposed in [119], which we refer to as the initial solution. The heuristic relies on spatial temperature vari-

Table 3.1: Results of the optimization procedure with respect to the number of processing elements, including the computational speed

n_p	n_t	MTTF (×)	Energy (×)	Time (s)
2	40	39.41	0.97	7.84
4	80	37.11	0.99	65.76
8	160	31.36	0.97	759.29
16	320	13.51	0.98	3484.59

Table 3.2: Results of the optimization procedure with respect to the number of tasks, including the computational speed

n_p	n_t	MTTF (×)	Energy (×)	Time (s)
4	40	64.53	0.88	9.96
4	80	38.01	0.96	56.57
4	160	18.08	1.07	352.20
4	320	12.92	1.05	408.42

ations and tries to minimize the maximum temperature while satisfying real-time constraints. This initial solution is also used to decide on the deadline constraint $\tau_{\max}$ in Equation 3.12: it is set to the duration of the initial schedule extended by 5%. Furthermore, the initial solution serves as a baseline for evaluating the performance of the solutions delivered by our optimization.

In the first set of experiments, we change the number of processing elements n_p while keeping the number of tasks n_t per processing element constant and equal to 20. For each problem, we generate 20 random task graphs and compute the average change in the MTTF relative to the initial solution. In addition, we calculate the average change in energy consumption. The results are reported in Table 3.1, which also shows the average time that is taken by the optimization procedure. It can be seen that the reliability-aware optimization increases the MTTF by a factor of 13–40. Even for large problems—such as those with 16 processing elements executing 320 tasks—a feasible schedule that significantly improves the lifetime of the system can be found in an affordable time. Moreover, as shown in Table 3.1, the optimization does not have much of an impact on the energy efficiency of the system.

In the second set of experiments, we keep the quad-core platform and vary the number of tasks n_t of the application. As before, for each problem, we generate 20 random task graphs and monitor the changes in the MTTF and energy consumption relative to the initial solution. The results can be seen in Table 3.2. The observations are similar to those made with respect to Table 3.1.

The experiments show that our optimization is able to effectively increase the MTTF of the system at hand. The efficiency is due to the fast and accurate approach to dynamic steady-state temperature analysis presented in Sec-

Table 3.3: Results of the optimization procedure with respect to the solution to dynamic steady-state temperature analysis

n_p	n_t	MTTF (×)	MTTF_a (×)	MTTF_b (×)
4	40	64.53	1.29	25.10
4	80	38.01	1.67	13.87
4	160	18.08	2.02	5.33
4	320	12.92	1.72	3.82

tion 3.4.2, which is at the heart of the optimization procedure. Due to its speed, the technique allows a large portion of the design space to be explored.

In order to illustrate the above aspect, let us replace our solution, inside the optimization framework, with (a) the one based on iterative transient analysis with HotSpot and (b) the one based on static steady-state analysis; see Section 3.4.1. The goal is to compare the results in Table 3.2 with the results produced by the two alternative methods when they are given the same amount of time as the amount taken by our method. For each problem, the optimized MTTF produced by either of the two approximate methods is re-evaluated using our exact method. The results are summarized in Table 3.3 where MTTF_a and MTTF_b stand for the two alternative methods, respectively. It can be seen that, for example, the lifetime of the platform running 160 tasks can be extended by a factor of 18 using our technique, whereas the best solutions found by the other two methods within the same time frame are only 2–5 times better than the initial solution. The reason for the poor performance of iterative transient analysis with HotSpot is the excessively long execution time of the calculation of dynamic steady-state temperature profiles, which means that this method allows for a very shallow exploration of the design space. In the case of static steady-state analysis, the reason is different: the method is fast but also very inaccurate, which is discussed and illustrated in Section 3.4.1. The inaccuracy drives the optimization toward solutions that turn out to be of low quality.

The experiments show that the impact of the optimization on energy consumption is insignificant. This is not surprising: the optimization searches toward low-temperature solutions, which are also implicitly the low-leakage ones. In order to explore this further, let us perform a multiobjective optimization along the dimensions of energy and reliability, and let us use NSGA-II [31] for this purpose. The resulting Pareto front averaged over 20 applications with 80 tasks deployed on a quad-core platform is displayed in Figure 3.8. It can be seen that the variation in the energy change is less than 2%. This means that the solutions optimized for the MTTF and the solutions optimized for energy have almost identical energy consumption values. At the same time, the difference along the MTTF dimension is huge. This implies that, by ignoring the reliability aspect, the designer might end up with a significantly decreased MTTF without any significant gain in terms of energy consumption.

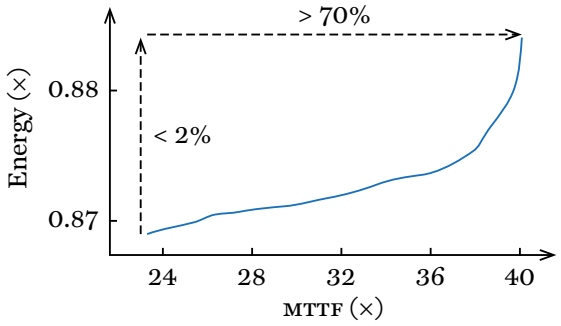

Figure 3.8: Pareto front delivered by the multiobjective optimization

Finally, we consider a real-life application, namely an MPEG-2 decoder [38], which is assumed to be deployed on a dual-core platform. The decoder is analyzed and split into 34 tasks. The parameters of each task are obtained through a system-level simulation using MPARM [5]. The deadline is set to 40 ms assuming 25 frames per second. The solution found by the proposed method improves the lifetime of the system by a factor of 23.59 with a 5% energy saving compared to the initial solution. The solutions found by undertaking the same optimization via the two alternative methods mentioned earlier are only 5.37 and 11.5 times better, respectively, than the initial one.

To conclude, the experimental results have demonstrated the superiority of the proposed approach to dynamic steady-state temperature analysis in the context of reliability optimization compared to the state of the art.

3.7 Conclusion

In this chapter, we have elaborated on deterministic analysis and design of electronic systems. Due to the paramount importance of power and temperature for efficiency and robustness, our focus has been primarily on these two quantities. Three types of temperature analysis have been covered in detail: transient analysis, static steady-state analysis, and dynamic steady-state analysis. We have proposed an auxiliary transformation of the temperature model that allows for computationally efficient and convenient calculations associated with temperature analysis. Leveraging the transformation, we have developed a fast and accurate solution to dynamic steady-state analysis. Using our solution, we have conducted a temperature-aware reliability optimization addressing thermal-cycling fatigue and have shown that taking temperature fluctuations into consideration can significantly prolong the lifetime of the system under consideration without affecting its energy efficiency.

As discussed in Section 1.2, uncertainty-unaware analysis misrepresents reality, and uncertainty-unaware design produces unsatisfactory, if not strictly

dangerous, products. Therefore, the designer of electronic systems cannot directly rely on techniques such as the ones presented in this chapter. These techniques, however, constitute an important building block for those that take uncertainty into consideration, which we discuss in the following chapters.

4 Analysis of Process Uncertainty

Starting with this chapter and relying on the exposition given in Chapter 3, we turn our attention to techniques that address uncertainty. In this particular chapter, we aim to analyze process variation across semiconductor wafers.

4.1 Introduction

As discussed in Section 1.1.1, process variation is an exigent concern of electronic-system designs, since it can lead to deterioration in efficiency and to faults of various magnitudes. Therefore, process variation should be adequately analyzed as the first step toward an efficient and robust product.

An important problem in this regard is to characterize the on-wafer distribution of a quantity deteriorated by process variation given a set of measurements. The problem belongs to the class of inverse problems, since the measured data can be seen as an output of the system at hand, and the desired quantity can be seen as an input. Such an inverse problem is addressed here.

Our goal is to estimate on-wafer distributions of arbitrary process parameters with high accuracy and at low costs. The goal is accomplished by measuring auxiliary quantities that are more convenient and less expensive to work with, and then employing statistics in order to infer the desired parameters from the measurements. Specifically, we propose a novel approach to the quantification of process variation based on indirect and potentially incomplete and noisy measurements. Moreover, we develop a solid framework around the proposed idea and perform a thorough study of various aspects of our technique.

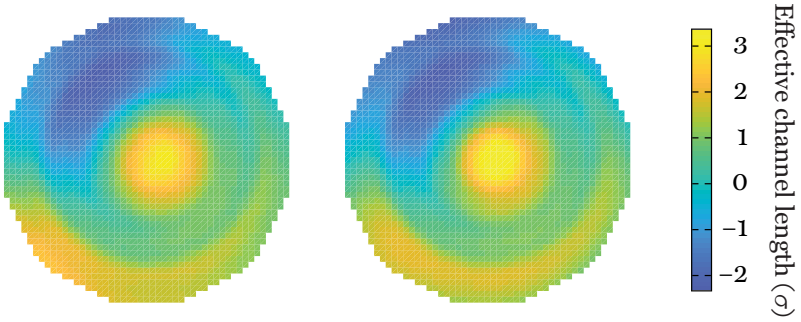

Figure 4.1: Example of a true distribution (*left*) and an inferred distribution (*right*) of the effective channel length across a silicon wafer

4.2 Motivational Example

Consider the distribution of the effective channel length, which we denote by g in this section, across a silicon wafer. As noted in Section 2.2, g has one of the strongest effects on the leakage current and consequently on power and temperature [58]. At the same time, g is well known to be severely affected by process variation [12, 103]. Therefore, the distribution of g is not uniform across the wafer. For concreteness, let this distribution be the one depicted on the left-hand side of Figure 4.1. The gradient from blue to yellow represents the transition of g from low to high values, and the scale is given in terms of the number of standard deviations away from the mean value; the exact experimental setup will be described in detail in Section 4.6. Hence, the blue regions have a high level of power consumption and heat dissipation.

Assume that the technological process imposes a lower bound $g_{\min}$ on g. It separates defective dies ($g < g_{\min}$) from those that are acceptable ($g \geq g_{\min}$). In order to reduce costs, the manufacturer is interested in detecting the faulty dies and taking them out of the production process at the earliest possible stage. The possible actions with respect to a die on the wafer are then (a) to keep the die if it conforms to the specification or (b) to dispose of it otherwise.

In order to analyze the variability in the effective channel length g across the wafer, one could remove the top layer of the dies (hence destroying them) and measure g directly. Alternatively, despite the fact that the knowledge of g is more preferable, one could step back and decide to quantify process variation using an auxiliary parameter h that can be measured without damaging the dies; for instance, h could be the leakage current. It should be noted that, in this second case, h is the final product of the analysis, and g remains unknown.

In either case, adequate test structures have to be present on the dies in order to take measurements at a sufficient number of locations with a sufficient level of granularity. Such a sophisticated test structure might not always be

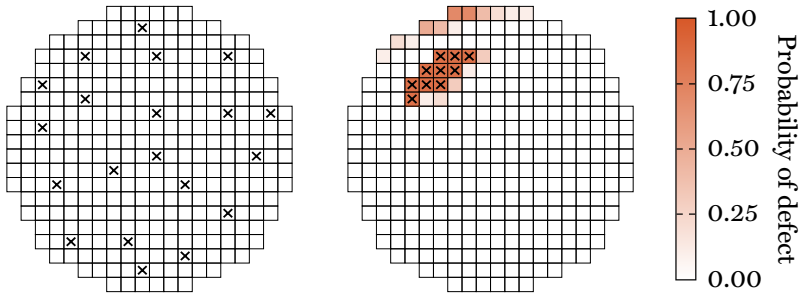

Figure 4.2: Locations of the measured dies (*left*) and the defective dies (*right*) as well as the inferred probability of defect (*right*)

readily available, and its deployment might significantly increase production expenses. Moreover, as noted earlier, the first approach implies that the measured dies have to be disposed of afterwards, and the second approach implies that further design decisions will be based on a surrogate quantity h instead of the primary target of process variation g, which could compromise these decisions. The latter concern is particularly prominent in situations where the production process is not yet completely stable, and design decisions based on the primary subjects of process variation are consequently preferable.

Our technique operates differently. In this example, in order to characterize the effective channel length g, we begin by measuring an auxiliary quantity h. The quantity is required to depend on g, and it can be chosen to be convenient from a measurement perspective. The distribution of g across the whole wafer is then obtained by inferring it from the collected measurements of h. Our technique permits these measurements to be taken only at a small number of locations on the wafer and to be corrupted by noise, which could be due to the imperfection of the measurement equipment that is utilized.

Let us consider one particular h that can be used to study the effective channel length g. Specifically, let h be temperature, which will be discussed further in Section 4.6. We can then apply a fixed workload—for instance, we can run the same application under the same conditions—to a few dies on the wafer and measure the corresponding temperature profiles. Since temperature does not require extra equipment to be deployed on the wafer and can be tracked using infrared cameras [78] or built-in components of the dies, our approach can reduce the costs associated with analysis of process variation. The results of our framework applied to a set of noisy temperature profiles measured only on 7% of the dies are shown on the right-hand side of Figure 4.1, and the locations of the measured dies are depicted on the left-hand side of Figure 4.2. It can be seen that the two maps in Figure 4.1 closely match each other, implying that the distribution of g is reconstructed with a high level of accuracy.

Another characteristic of the proposed framework is that probabilities of various events, for instance, $\mathbb{P}(g \geq g_{\min})$, can be readily estimated. This is important, since the true values are unknown in reality; otherwise, we would not need to infer them. Therefore, we can rely on our decisions only up to a certain probability. We can then reformulate the decision rule given earlier as follows: (a) to keep the die if $\mathbb{P}(g \geq g_{\min})$ is larger than a certain threshold or (b) to dispose of it otherwise. An illustration of following this rule is given on the right-hand side of Figure 4.2 where $g_{\min}$ is set to two standard deviations below the mean value of g, the probability threshold of the first action is set to 0.9, the crosses mark both the true and inferred defective dies (they coincide), and the gradient from white to orange corresponds to the inferred probability of defect. It can be seen that the inference accurately detects the faulty dies.

In addition, we can introduce a trade-off action between action (a) and action (b) as follows: (c) to expose the die to a thorough inspection (for instance, via a test structure) if $\mathbb{P}(g \geq g_{\min})$ is smaller than the threshold of action (a) but larger than another threshold. For instance, action (c) can be taken if $0.1 < \mathbb{P}(g \geq g_{\min}) < 0.9$. In this scenario, we can reduce expenses by examining only those dies for which there is no sufficiently strong evidence of their satisfactory or unsatisfactory condition. Furthermore, one can take into consideration a so-called utility function, which, for each combination of an outcome of g and an action that is taken, returns the gain that the decision maker obtains. For example, such a function could favor a rare omission of malfunctioning dies over a frequent inspection of correct dies, as the latter might involve much higher costs. The optimal decision is given by the action that maximizes the expected utility with respect to both the observed data and prior knowledge regarding g. Consequently, all possible values of g weighted by their probabilities are taken into account in the final decision while also incorporating the preferences of the designer via the utility function.

4.3 Problem Formulation

Consider a generic silicon wafer that accommodates n_d dies. We are interested in studying a certain process parameter, which is denoted by g and referred to as the quantity of interest. Due to process variation, the value of g deviates from the nominal one, and it can be different at different locations on the wafer. Direct measurement of this parameter is assumed to be impractical.

The goal is to develop a framework for identifying the on-wafer distribution of the quantity of interest g with the following properties: (a) low measurement costs, (b) robustness to measurement noise, (c) ability to accommodate prior knowledge about g, (d) ability to assess the credibility of the collected data and the corresponding predictions, and (e) high computation speed.

4.4 Previous Work

There are a number of related studies that we would like to highlight. Bayesian inference, which is briefly introduced in Appendix A.3, is utilized in [123] for identifying the optimal set of locations on the wafer where the parameter under consideration should be measured in order to characterize it with the maximal accuracy. The expectation-maximization algorithm is considered in [93] in order to estimate missing test measurements. In [84], the authors consider an inverse problem focused on the inference of power consumption based on transient temperature maps by means of Markov random fields. Another temperature-based characterization of power is developed in [78] where a genetic algorithm is employed for the reconstruction of the power model.

It should be noted that the procedures proposed in [93, 123] operate on direct measurements, meaning that the output is the same quantity as the one being measured. In particular, these procedures rely heavily on the availability of adequate test structures on the dies and are practical only for secondary quantities affected by process variation, such as delays and currents, but not for the primary ones, such as various physical dimensions. Consequently, they often lead to excessive costs and have a limited range of applicability. On the other hand, the approaches given in [78, 84], which concentrate on the power consumption of a single die, are not concerned with process variation.

4.5 Proposed Solution

In order to achieve the established goal, we make use of indirect measurements. Specifically, instead of g, we measure an auxiliary parameter h, which we refer to as the quantity of measurement. The observations of h are then processed via Bayesian inference (see Appendix A.3) in order to derive the on-wafer distribution of g. The quantity h is chosen such that (a) h is convenient and cheap to be measured; (b) h depends on g, which is signified by $h = f(g)$; and (c) there is a way to compute h given g. The last requirement means that f should be known. However, f does not have to be specified analytically, since our framework treats f as a "black box." For example, f can be a piece of code.

Without loss of generality, we adhere to the following convention. Each die is a potential measurement site, and $\hat{n}_d < n_d$ denotes the number of those sites that are actually measured. Each site comprises n_p measurement points, and there are n_s data instances per point. For instance, in the example given in Section 4.2, an observation at a site is a temperature profile $\mathbf{Q}$, which is a matrix capturing the temperatures of n_p processing elements at n_s moments in time as defined in Equation 2.2. Denote the collected data by H and assume that the locations of the measurement points are recorded along with H.

It is worth noting that, if f is the identity function (that is, $h = f(g) = g$), the proposed technique focuses primarily on the reconstruction of any missing

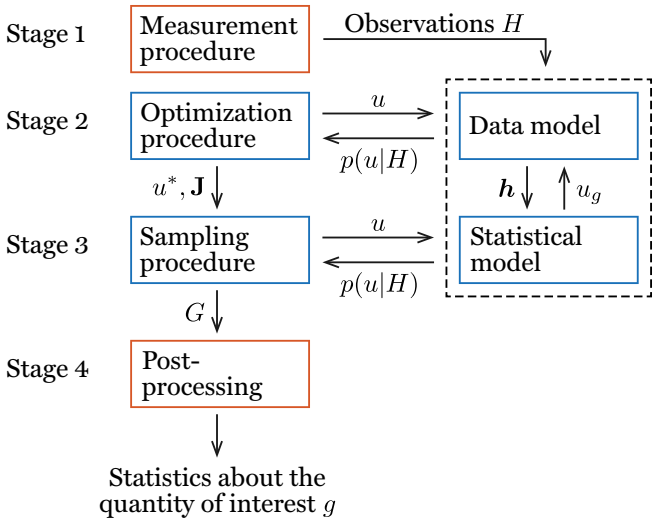

Figure 4.3: Overview of the proposed solution for characterizing process variation across silicon wafers

observations in H, that is, on the unobserved sites on the wafer. From this standpoint, our approach is a generalization of those developed in [93, 123].

In the rest of this section, we present our framework for characterizing process variation. The technique revolves around Bayes's theorem [42]

$$p(u|H) \propto p(H|u)p(u). \tag{4.1}$$

In our context, the theorem relates the posterior density function $p(u|H)$ of u given H with the likelihood function $p(H|u)$ of H given u and the prior density function $p(u)$ of u; these concepts are introduced in Appendix A.3, and they will be discussed further in the following. The framework is divided into four major stages depicted in Figure 4.3. Stage 1 is the data-harvesting stage where the designer collects a set of observations of the quantity of measurement h, thereby forming the data set H. At Stage 2, we perform an optimization procedure that is designed to assist the sampling procedure at Stage 3. The latter produces a collection of samples of the quantity of interest g, such as the effective channel length, denoted by G. This data set G is then processed at Stage 4 in order to estimate the desired statistics about g, such as the probability of g being smaller than a certain threshold; recall the example in Section 4.2.

It can be seen in Figure 4.3 that Stage 2 and Stage 3 actively communicate with the two models shown on the right-hand side, which are called the data model and statistical model. We begin by elaborating on these models.

4.5.1 Data Model

The data model relates the quantity of interest g with the quantity of measurement h, which is denoted as follows:

$$h = f(g).$$

The function f depends on the choice of h and is specified by the designer. The data model is utilized in order to predict the values of h at the same measurement sites, at the same measurement points, and with the same number and meaning of data instances as the ones in H obtained at Stage 1. The resulting data are stacked into a single vector denoted by $\boldsymbol{h} \in \mathbb{R}^{\hat{n}_d n_p n_s}$. Let also $\hat{\boldsymbol{h}} \in \mathbb{R}^{\hat{n}_d n_p n_s}$ be a stacked version of the data in H such that the respective elements of $\boldsymbol{h}$ and $\hat{\boldsymbol{h}}$ correspond to the same locations on the wafer.

In order to acquire a better understanding of the data model, let us return to the example given in Section 4.2. In this case, g stands for the effective channel length, and h for the temperature profile $\mathbf{Q}$ corresponding to a fixed workload. The data model can be roughly divided into two transitions: (a) from the effective channel length g to the static power consumed by the platform at hand and (b) from this static power to the corresponding temperature profile h. At this point, it is worth recalling the power model presented in Section 2.2. The first transition is due to the dependence of the leakage current on the effective channel length, which is implicitly present in Equation 2.4. The transition can then be made by means of the model shown in Equation 2.4 or any of its variations; see, for instance, [12, 58, 103, 121]. In particular, an adequate model of static power can be constructed via fitting to SPICE simulations of reference electrical circuits. The only requirement for such a model is that it should be parameterized by g. In addition, it can be parameterized by temperature in order to take account of the power-temperature interdependence described in Section 2.2. The second transition is made by combining the static power component with the dynamic power that corresponds to the workload being considered. The resulting total power and the relevant temperature-related information—such as the floorplan and thermal parameters of the platform—are fed to a temperature simulator in order to acquire the corresponding temperature h, which is discussed in detail in Chapter 3.

4.5.2 Statistical Model

Once the wafer has been fabricated, the values of g across the wafer are fixed; however, they remain unknown to the designer. In order to infer them, we employ the model developed in the current subsection, which can also be seen in Figure 4.4. The development consists of the five steps described below.

Step 1 is to assign an adequate model to the unknown g. We model g as a Gaussian process [92], since (a) it is flexible in capturing the variation patterns induced by the fabrication process, (b) it is computationally efficient, and

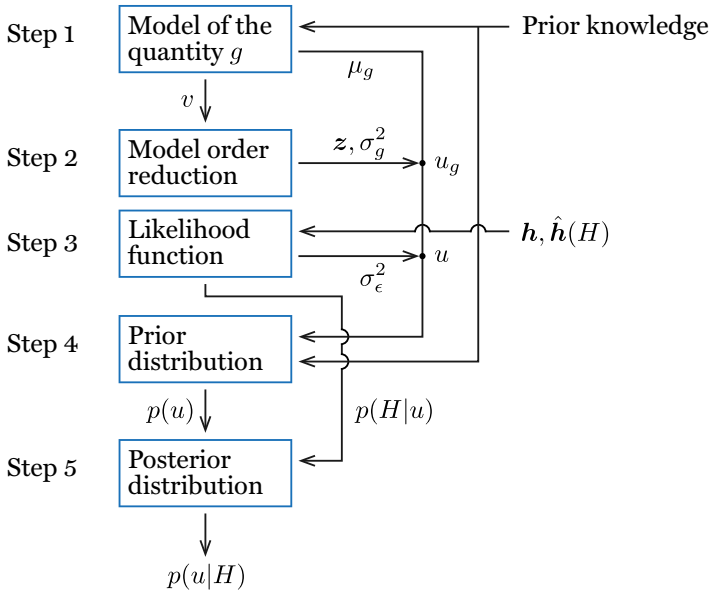

Figure 4.4: Statistical model of the proposed solution for characterizing process variation across silicon wafers

(c) Gaussian distributions are often adequate models of uncertainty due to process variation [58, 93, 103]. The model is denoted by

$$g|u_g \sim \text{Gaussian Process}(\mu, v) \tag{4.2}$$

where $\mu : \mathbb{R}^2 \to \mathbb{R}$ and $v : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ are the mean function and covariance function of g, respectively, which take locations on the wafer as arguments. The notation also indicates that g depends on a set of parameters u_g, which we shall identify later on. Prior to taking any measurements, g is assumed to be spatially unbiased. Therefore, we let μ be a single location-independent parameter μ_g, which means that $\mu(\boldsymbol{r}) = \mu_g$ for any $\boldsymbol{r} \in \mathbb{R}^2$. The covariance function v is chosen to be the following:

$$v(\boldsymbol{r}_1, \boldsymbol{r}_2) = \sigma_g^2 \, k(\boldsymbol{r}_1, \boldsymbol{r}_2) \tag{4.3}$$

for $\boldsymbol{r}_1 \in \mathbb{R}^2$ and $\boldsymbol{r}_2 \in \mathbb{R}^2$ where

$$k(\boldsymbol{r}_1, \boldsymbol{r}_2) = w \, k_{\text{SE}}(\boldsymbol{r}_1, \boldsymbol{r}_2) + (1 - w) k_{\text{OU}}(\boldsymbol{r}_1, \boldsymbol{r}_2) \tag{4.4}$$

is the correlation function, and

$$k_{\text{SE}}(\boldsymbol{r}_1, \boldsymbol{r}_2) = \exp\left(-\frac{\|\boldsymbol{r}_1 - \boldsymbol{r}_2\|_2{}^2}{\ell_{\text{SE}}^2} \right) \text{ and}$$
$$k_{\text{OU}}(\boldsymbol{r}_1, \boldsymbol{r}_2) = \exp\left(-\frac{|\|\boldsymbol{r}_1\|_2 - \|\boldsymbol{r}_2\|_2|}{\ell_{\text{OU}}} \right)$$

are the squared-exponential and Ornstein–Uhlenbeck kernels [92], respectively. In these formulae, σ_g^2 is the variance of g, $w \in [0, 1]$ is a weight coefficient, $\ell_{\mathrm{SE}} > 0$ and $\ell_{\mathrm{OU}} > 0$ are so-called length-scale parameters, and $\|\cdot\|_2$ stands for the Euclidean distance. The choice of v is based on the observations of the correlation structures induced by the fabrication process [12, 15]. Specifically, k_{SE} imposes similarities on locations that are close to each other on the wafer, and k_{OU} imposes similarities on locations that are at the same distance from the center of the wafer. The parameters ℓ_{SE} and ℓ_{OU} control the extent of these similarities, that is, the range where the correlation between two locations is significant. Although all the above parameters of g can be inferred from data, for simplicity, we focus only on μ_g and σ_g^2. The rest of the parameters—namely w, ℓ_{SE}, and ℓ_{OU}—are assumed to be determined prior to our analysis based on knowledge of the variation patterns that are typical for the fabrication process being considered; see [77] and the references therein.

Step 2 is to make the above model of g computationally tractable. The model is an infinite-dimensional object, as it characterizes a continuum of locations. For practical computations, it should be reduced to a finite-dimensional one. First, g is discretized with respect to the union of two sets of locations. The first one is composed of the $\hat{n}_d n_p$ points where the observations in H are made ($\hat{n}_d$ measurement sites with n_p measurement points each), and the other one of the locations where the designer wishes to characterize g. For simplicity, assume that the designer is interested in all the sites, which results in $n_d n_p$ locations in total. Let $\boldsymbol{g} \in \mathbb{R}^{n_d n_p}$ store the values of g at these locations. Second, the dimensionality of the problem is reduced further via the Karhunen–Loève (KL) decomposition, which is introduced in Appendix A.4. Concretely, we perform the transformation shown in Equation A.7 with respect to the correlation matrix of $\boldsymbol{g}$ computed via Equation 4.4. The result is

$$\boldsymbol{g} = \mu_g \mathbf{1} + \sigma_g \mathbf{U} \tilde{\boldsymbol{\Lambda}}^{\frac{1}{2}} \boldsymbol{z} \tag{4.5}$$

where $\mathbf{1} = (1, \ldots, 1) \in \mathbb{R}^{n_d n_p}$, and $\boldsymbol{z} = (z_i) \in \mathbb{R}^{n_z}$ is a vector of independent random variables that obey the standard Gaussian distribution. The number n_z is the final dimensionality of the model of g; typically, $n_z \ll n_d n_p$. In addition, the parameters u_g in Equation 4.2 are now known: $u_g = \{\boldsymbol{z}, \mu_g, \sigma_g^2\}$; see Figure 4.4. The model is now ready for practical computations.

Step 3 is to specify the likelihood function in Equation 4.1, which is where the observed information is taken into account; see Appendix A.3. In our case, the observed information is the measurements H stacked into $\hat{\boldsymbol{h}}$ as described in Section 4.5.1. Since the measurement process is not perfect, we also have to take measurement noise into consideration. To this end, the observed $\hat{\boldsymbol{h}}$ is assumed to deviate from the data model's prediction $\boldsymbol{h}$ as follows:

$$\hat{\boldsymbol{h}} = \boldsymbol{h} + \boldsymbol{\epsilon}$$

where $\boldsymbol{\epsilon}$ is an $\hat{n}_d n_p n_s$-dimensional vector of noise, which is typically assumed to be white Gaussian noise [77, 92]. Without loss of generality, the noise is

assumed to be independent of g and to have the same magnitude for all measurements. Hence, the model of the noise is as follows:

$$\boldsymbol{\epsilon}|\sigma_\epsilon^2 \sim \text{Gaussian}(\mathbf{0}, \sigma_\epsilon^2 \mathbf{I})$$

where σ_ϵ^2 is the variance of the noise. At this point, all the parameters of the inference are identified, and they are $u = u_g \cup \{\sigma_\epsilon^2\} = \{\boldsymbol{z}, \mu_g, \sigma_g^2, \sigma_\epsilon^2\}$; see Figure 4.4. Taking the above into account, we obtain

$$\hat{\boldsymbol{h}}|u \sim \text{Gaussian}(\boldsymbol{h}, \sigma_\epsilon^2 \mathbf{I}). \tag{4.6}$$

The density function of this distribution is the likelihood $p(H|u)$ of our statistical model, which is the first element of the posterior in Equation 4.1.

Step 4 is to decide on the second element of the posterior in Equation 4.1, that is, on the prior $p(u)$. We put the following priors on the parameters in u:

$$\begin{aligned} \boldsymbol{z} &\sim \text{Gaussian}(\mathbf{0}, \mathbf{I}), \\ \mu_g &\sim \text{Gaussian}(\mu_0, \sigma_0^2), \\ \sigma_g^2 &\sim \text{Scaled Inverse } \chi^2(n_{ug}, \tau_g^2), \text{ and} \\ \sigma_\epsilon^2 &\sim \text{Scaled Inverse } \chi^2(n_{u\epsilon}, \tau_\epsilon^2). \end{aligned} \tag{4.7}$$

The prior of $\boldsymbol{z}$ is due to the decomposition in Equation 4.5. The other three priors, that is, a Gaussian and two scaled inverse chi-squared distributions, are a common choice for a Gaussian model whose mean and variance are unknown. The parameters μ_0, τ_g^2, and τ_ϵ^2 represent the presumed values of μ_u, σ_g^2, and σ_ϵ^2, respectively, and are set by the designer based on prior knowledge of the technological process and measurement equipment. The parameters σ_0, n_{ug}, and $n_{u\epsilon}$ reflect the precision of this information. When the prior knowledge is weak, less specific priors can be considered [42]. Finally, $p(u)$ in Equation 4.1 is obtained by multiplying the densities of the priors in Equation 4.7.

Step 5 is to calculate the posterior $p(u|H)$ in Equation 4.1. To this end, the likelihood function $p(H|u)$, which is the density of the distribution shown in Equation 4.6, and the prior $p(u)$, which is the product of the densities of the distributions shown in Equation 4.7, are put together. The density of the resulting posterior distribution is as follows:

$$p(u|H) \propto p(\hat{\boldsymbol{h}}|\boldsymbol{z}, \mu_g, \sigma_g^2, \sigma_\epsilon^2) p(\boldsymbol{z}) p(\mu_g) p(\sigma_g^2) p(\sigma_\epsilon^2). \tag{4.8}$$

Provided that there is a way to draw samples from Equation 4.8, g can be readily analyzed, as we shall see in Section 4.5.5. The problem, however, is that direct sampling of the posterior is difficult due to the data model involved in the likelihood function via $\boldsymbol{h}$; see Equation 4.6. In order to circumvent this problem, we utilize Markov chain Monte Carlo sampling and, more specifically, the Metropolis–Hastings algorithm [42], which is outlined in Appendix A.3. The algorithm operates on an auxiliary distribution called the proposal distribution. The construction of an adequate proposal is discussed next.

4.5.3 Optimization Procedure

In this subsection, we describe the objective of Stage 2 in Figure 4.3. Although the requirements for the proposal distribution are mild, it is often difficult to pick an efficient proposal, that is, a proposal that would yield a good approximation with as few evaluations of the posterior—and thus of the data model in Section 4.5.1—as possible. This choice is especially difficult in high-dimensional problems, and our problem—involving around 30 parameters, as we shall see in Section 4.6—is one them. Therefore, a careful construction of the proposal is an essential component of the proposed framework.

A common technique for constructing a high-quality proposal is to optimize the posterior given in Equation 4.8. Specifically, we seek a value u^* of u that maximizes Equation 4.8 and hence has the maximum posterior probability. In addition, we calculate the negative of the Hessian matrix at u^*, which is called the observed information matrix and denoted by $\mathbf{J}$; see the output of Stage 2 in Figure 4.3. Using u^* and $\mathbf{J}$, we can construct a proposal that allows the Metropolis–Hastings algorithm (a) to start producing samples directly from the desired regions of high probability and (b) to explore those regions more rapidly. The usage of u^* and $\mathbf{J}$ is explained next.

4.5.4 Sampling Procedure

Let us turn to Stage 3 in Figure 4.3. In order to construct an adequate proposal and utilize it for sampling, we have at our disposal u^* and $\mathbf{J}$ from Stage 2. A commonly used proposal is a multivariate Gaussian distribution where the mean is the current location of the chain of samples started at u^*, and the covariance matrix is the inverse of $\mathbf{J}$ [42]. In order to speed up the sampling process, we would like to make use of parallel computing. The aforementioned proposal, however, is purely sequential, since the mean for the next sample draw is dependent on the previous sample. Therefore, we appeal to a variant of the Metropolis–Hastings algorithm known as the independence sampler [42]. In this case, a typical choice of the proposal is a multivariate t-distribution independent of the current position of the chain as follows:

$$u \sim t_{n_u}(u^*, \alpha^2 \mathbf{J}^{-1}) \tag{4.9}$$

where u^* and $\mathbf{J}$ are as in Section 4.5.3, n_u is the number of degrees of freedom, and α is a tuning constant controlling the standard deviation of the proposal. Now sampling the proposal in Equation 4.9 and evaluating the posterior in Equation 4.8 can be done in parallel. The obtained samples are then accepted or rejected as in the usual Metropolis–Hastings algorithm.

Having completed the sampling procedure, we obtain a collection of samples of the parameterization $u = \{\boldsymbol{z}, \mu_g, \sigma_g^2\}$. The first portion of the samples is typically discarded as being unrepresentative; this portion is known as the

burn-in period. The preserved samples of u are then passed through Equation 4.5 in order to compute samples of g, which are $n_d n_p$-dimensional. Denote the corresponding set of samples by G and let its cardinality be n_ω.

4.5.5 Post-Processing

At Stage 4 in Figure 4.3, using G, the designer computes the desired statistics about the quantity of interest g, such as the most probable value of g at some location on the wafer and the probability of a certain area on the wafer being defective. These computations are performed in the same way as it is typically done when Monte Carlo (MC) sampling is utilized. Specifically, they reduce to the estimation of expected values with respect to the posterior distribution of u given in Equation 4.8: in order to calculate a certain quantity dependent on g, one evaluates it for each sample in G and then takes the average value.

The strength of the Bayesian approach to inference starts to shine when one is interested in assessing the trustworthiness of the measured data and thus the credibility of estimates and decisions based on these data. Such an assessment can be readily undertaken using our framework, since the delivered posterior contains all the necessary information about the quantity of interest g. As discussed in Section 4.2, this is especially helpful in decision-making.

4.6 Experimental Results

In this section, we assess our framework for characterizing process variation presented in Section 4.5. All the experiments are conducted on a GNU/Linux machine equipped with an Intel Core i7 2.66 GHz and 8 GB of RAM. All the configuration files used in the experiments are available online at [17].

Our goal is to infer the effective channel length g from temperature h. Such a high-level parameter as temperature constitutes a challenging task for the inference of such a low-level parameter as the effective channel length, which implies a rigorous assessment of the proposed technique. The performance of our approach is expected only to increase when the auxiliary parameter h is closer to the target parameter g with respect to the data model $h = f(g)$ described in Section 4.5.1. For instance, such a closer quantity h could be the leakage current, but this might not always be the most preferable parameter to measure. Lastly, let us note that the chosen effective channel length is an important target, as it is strongly affected by process variation and considerably impacts power consumption and heat dissipation [12, 58, 59, 103]. It also affects other process-related parameters, such as the threshold voltage.

We first describe the default configuration, which will be adjusted later on according to the purpose of each particular experiment. We consider a 45-nm technological process. The diameter of the wafer is 20 dies, and the total number of dies n_d is 316. The number of measured dies $\hat{n}_d$ is 20, and these dies are chosen by an algorithm that strives for even coverage of the wafer. The

fabricated platform has four processing elements, and they are the points at which measurements are taken; that is, $n_p = 4$. The floorplan of the platform is constructed in such a way that the processing elements form a regular grid. The dynamic power profiles involved in the experiments are based on simulations of applications randomly generated by TGFF [34]. The model of static power parameterized by temperature and the effective channel length is constructed by fitting to SPICE simulations of reference electrical circuits that are composed of BSIM4 devices [106] configured according to the 45-nm PTM HP model [108]. The construction of thermal RC circuits is delegated to HotSpot [100], and temperature analysis is performed as described in Section 3.2.2. The sampling interval of power and temperature profiles is 1 ms.

The input data set H is obtained as follows: (a) draw a sample of g from a Gaussian distribution with a mean of 17.5 nm (in accordance with the technological process under consideration [108]) and a covariance function equal to Equation 4.3 with a standard deviation of 2.25 nm; (b) perform one fine-grained temperature simulation for each of the $\hat{n}_d$ dies selected for measurement; (c) thin the obtained temperature profiles so that each has only n_s, which is 20 by default, evenly spaced moments in time; and (d) perturb the resulting data using white Gaussian noise with a standard deviation of 1°C.

Let us turn to the statistical model in Section 4.5.2 and summarize the intuition behind the model's parameters and the process by which they are assigned. In the correlation function given in Equation 4.4, the weight parameter w and the two length-scale parameters ℓ_{SE} and ℓ_{OU} should be set according to the variation patterns that are typical for the fabrication process at hand [12, 15]; we set w to 0.7 and ℓ_{SE} and ℓ_{OU} to half the radius of the wafer. The threshold parameter η of the model-order-reduction procedure shown in Equation A.6 and utilized in Equation 4.5 should be set high enough in order to preserve a sufficiently large portion of the variance of the data, thereby keeping the model sufficiently accurate; we set it to 0.99. The resulting dimensionality n_z of $\boldsymbol{z}$ in Equation 4.5 is found to be 27 or 28. The parameters μ_0 and τ_g of the prior given in Equation 4.7 are specific to the technological process under consideration; we set μ_0 to 17.5 nm and τ_g to 2.25 nm. The parameters σ_0 and n_{ug} used in Equation 4.7 determine the precision of the information about μ_0 and τ_g and are set according to the beliefs of the designer; we set σ_0 to 0.45 nm and n_{ug} to 10. The latter can be thought of as the number of imaginary observations that the choice of τ_g is based on. The parameter τ_ϵ in Equation 4.7 represents the precision of the equipment utilized for collecting H and can be found in the technical specification of that equipment; we set τ_ϵ to 1°C. The parameter $n_{u\epsilon}$ in Equation 4.7 has the same interpretation as n_{ug}; we set it to 10 as well. In Equation 4.9, n_u and α are tuning parameters, which are configured based on experiments; we set n_u to 8 and α to 0.5. The number of sample draws is another tuning parameter, which we set to 10^4. The first half of these samples is ascribed to the burn-in period mentioned in Appendix A.3, and the second one constitutes G; in this case, $n_\omega = 5 \times 10^3$. In the

Table 4.1: Computational speed and accuracy of the proposed solution with respect to the number of measurement sites

Number of sites	1	10	20	40	80	160
Optimization (min)	0.41	2.49	3.34	4.59	7.33	10.29
Sequential (min)	2.40	3.99	4.60	5.79	8.49	12.96
Total (min)	2.81	6.47	7.94	10.38	15.81	23.25
Parallel (min)	0.61	1.02	1.18	1.51	2.16	3.62
Total (min)	1.02	3.50	4.52	6.10	9.49	13.91
NRMSE (%)	30.49	4.40	3.42	1.09	0.85	0.67

optimization described in Section 4.5.3, we use the quasi-Newton algorithm [87]. For parallel computations, we utilize four computational cores.

In order to ensure that the experimental setup is adequate, we first perform a detailed inspection of the results obtained for one particular example with the default configuration. The true and inferred distributions of the quantity of interest are shown in Figure 4.1 where the normalized root-mean-square error (NRMSE) is below 2.8%, and the absolute error is bounded by 1.4 nm, which suggests that our framework produces a close match to the true value of the quantity. We also investigate the behavior of the constructed Markov chains and the quality of the proposal distribution. All the observations indicate that the optimization and sampling procedures are properly configured.

In the following subsections, we consider the experimental setup described above and alter a single parameter at a time in order to investigate its impact. Specifically, we change (a) the number of measurement sites $\hat{n}_d$, which is 20 by default; (b) the number of measurement points per site n_p, which is 4 by default; (c) the number of data instances per point n_s, which is 20 by default; and (d) the standard deviation of measurement noise σ_ϵ, which is 1°C by default.

4.6.1 Number of Measurement Sites

Let us vary the number of measured sites or dies $\hat{n}_d$. The scenarios being considered are 1, 10, 20, 40, 80, and 160 dies. The obtained results are shown in Table 4.1. In this and the following tables, we report the optimization time and sampling time separately, which correspond to Stage 2 and Stage 3 in Figure 4.3, respectively. In addition, the sampling time is given for two cases: sequential and parallel computing, which is followed by the total time and resulting error. The computation time of the post-processing stage, Stage 4, is not given, as it is negligibly small. The sequential sampling time is the most representative indicator of the computational complexity of the proposed framework, since the number of samples is always fixed, and there is no parallelization. Therefore, we refer to this value in most of the discussions given below.

Table 4.2: Computational speed and accuracy of the proposed solution with respect to the number of measurement points

Number of points	2	4	8	16	32
Optimization (min)	2.67	3.34	5.20	7.37	13.85
Sequential (min)	3.71	4.60	6.03	8.92	14.77
Total (min)	6.38	7.94	11.23	16.29	28.62
Parallel (min)	0.98	1.18	1.58	2.51	5.30
Total (min)	3.65	4.52	6.78	9.88	19.15
NRMSE (%)	4.71	3.42	3.68	2.73	1.94

It can be seen in Table 4.1 that the more data the proposed framework needs to process, the longer the execution time becomes, which is reasonable. The trend, however, is modest: when $\hat{n}_d$ is doubled, the computation time increases by less than a factor of two. Regarding accuracy, the error decreases definitively and drops below 4% when around 20 sites are measured, which is only 6%–7% of the total number of dies on the wafer under consideration.

4.6.2 Number of Measurement Points

In this subsection, we consider five platforms with different numbers of processing elements or, equivalently, measurement points n_p. The scenarios being considered are 2, 4, 8, 16, and 32 processing elements. The results are summarized in Table 4.2. The computation time grows along with n_p. This is expected, since the granularity of the temperature model is bound to the number of processing elements: each processing element contributes four thermal nodes to the thermal RC circuit that temperature analysis is based on; recall Section 2.3. Hence, temperature analysis becomes more expensive. Nevertheless, even for large examples, taking into account the complexity of the inference procedure and the yielded accuracy, the time requirement is readily acceptable. An interesting observation can be made with respect to the NRMSE: the error tends to decrease as n_p grows. The reason is that, with each measurement point, H delivers more information for the inference to work with.

4.6.3 Number of Data Instances

Now we change the number of data instances n_s, which, in this case, is the number of moments in time captured by temperature profiles. The scenarios being considered are 1, 10, 20, 40, 80, and 160 moments. The results are aggregated in Table 4.3. It can be seen that the growth in computation time is relatively small. One might expect this growth due to n_s to be the same as the one due to n_p, since, technically, the influence of n_p and n_s on the dimensionality of H

Table 4.3: Computational speed and accuracy of the proposed solution with respect to the number of data instances

Number of instances	1	10	20	40	80	160
Optimization (min)	1.12	3.02	3.34	3.62	3.64	4.20
Sequential (min)	2.40	4.38	4.60	4.67	4.80	4.97
Total (min)	3.52	7.40	7.94	8.29	8.44	9.16
Parallel (min)	0.62	1.13	1.18	1.22	1.25	1.30
Total (min)	1.74	4.16	4.52	4.84	4.89	5.50
NRMSE (%)	7.48	2.72	3.42	1.83	2.34	1.32

Table 4.4: Computational speed and accuracy of the proposed solution with respect to the standard deviation of noise

Deviation of noise (°C)	0.00	0.50	1.00	2.00
Optimization (min)	5.08	3.73	3.34	3.19
Sequential (min)	4.76	4.70	4.60	4.71
Total (min)	9.84	8.43	7.94	7.90
Parallel (min)	1.19	1.17	1.18	1.18
Total (min)	6.27	4.91	4.52	4.37
NRMSE (%)	0.02	2.71	3.42	4.05

is identical; recall that $\hat{\boldsymbol{h}} \in \mathbb{R}^{\hat{n}_d n_p n_s}$. However, the meanings of n_p and n_s are completely different, and hence the ways they manifest themselves in the inference algorithm are also different, which explains the discordant figures shown in Table 4.2 and Table 4.3. The NRMSE in Table 4.3 has a decreasing trend; however, this trend is less steady than the ones noted before. The finding can be explained as follows. The temporal distribution of the moments in time that are present in H changes, since these moments are kept evenly spaced across the time spans of the corresponding applications. Some moments can be more informative than others. Consequently, more or less representative samples can be accumulated in H, helping or misleading the inference. Additionally, we can conclude that a larger number of spatial measurements is more advantageous than a larger number of temporal measurements.

4.6.4 Deviation of Measurement Noise

In this subsection, we vary the standard deviation of measurement noise, which corrupts H. The cases being considered are 0°C, 0.5°C, 1°C, and 2°C [78]. Note that the corresponding prior in Equation 4.7 is kept unchanged. The results are given in Table 4.4. It can be seen that the sampling time is ap-

proximately constant. However, we observe an increase in the optimization time when the level of noise decreases, which can be ascribed to greater opportunities for perfection for the optimization procedure. Another observation revealed by this experiment is that, in spite of the fact that the inference operates on indirect and incomplete data, a thoroughly calibrated piece of equipment can considerably improve the quality of prediction. However, even with a noise of 2°C—meaning that measurements are dispersed over a wide band of 8°C with a probability of more than 0.95—the NRMSE is still only 4%.

4.6.5 Sequential and Parallel Sampling

Lastly, we elaborate on the sequential and parallel sampling strategies. In the sequential Metropolis–Hastings algorithm, the optimization time is typically smaller than the time needed for drawing posterior samples. The situation changes when parallel computing is utilized. When four cores are working in parallel, the sampling time decreases by a factor of 3.81 on average, which indicates good parallelization properties of the chosen sampling strategy. The overall speedup ranges from 1.49 to 2.75 with an average value of 1.77, which can be pushed even further by employing more computational cores.

4.7 Conclusion

In this chapter, we have presented a framework for characterizing process variation across semiconductor wafers based on indirect measurements. The technique has been exposed to extensive experiments, and the obtained results have shown the computational efficiency and accuracy of our approach.

The presented framework is capable of quantifying primary parameters affected by process variation, such as the effective channel length, which is in contrast to the former techniques where only secondary parameters are considered, such as the leakage current. Instead of taking direct measurements of the quantity of interest, we employ Bayesian inference in order to draw justifiable conclusions based on indirect observations, such as temperature measurements. Our approach has low costs, since it need not require deployment of expensive test structures on the dies, and in scenarios where such structures are already available, it might require engaging only a small subset of them.

Finally, we would like to emphasize that, although the proposed framework has been demonstrated by considering the effective channel length and temperature, it can be readily utilized for analyzing any other quantity of interest based on any other quantity of measurement provided that the latter depends on the former, and that a model of this dependence is available.

5 Analysis and Design under Process Uncertainty

In this chapter, we shift our attention from characterizing process variation to analyzing its destructive implications for high-level characteristics of electronic systems so that these implications can be taken into consideration.

5.1 Introduction

As discussed in Section 1.1.1, process variation has to be addressed by the designer. In order to assist the designer, we present a framework that allows one to propagate uncertainty stemming from process variation through the system under development and thereby investigate and take account of its impact on the system's behavior. In particular, it enables power and temperature analysis as well as reliability analysis to be performed in such a way that process variation is adequately considered. In the case of reliability analysis, for example, our framework delivers a reliability function—founded on the basis of well-established reliability models—with a computationally efficient probabilistic parameterization. The proposed approach is demonstrated by considering a number of problems. In particular, we analyze systems with periodic workloads that suffer from thermal cycling and construct and execute a design-space-exploration procedure aimed at minimizing the expected energy consumption of the system under a number of probabilistic constraints.

5.2 Motivational Example

Consider a quad-core architecture whose leakage current is uncertain to the designer due to its dependence on a number of process parameters that are affected by process variation. Assume first that these parameters have nominal values. We can then simulate the system under a certain workload and observe

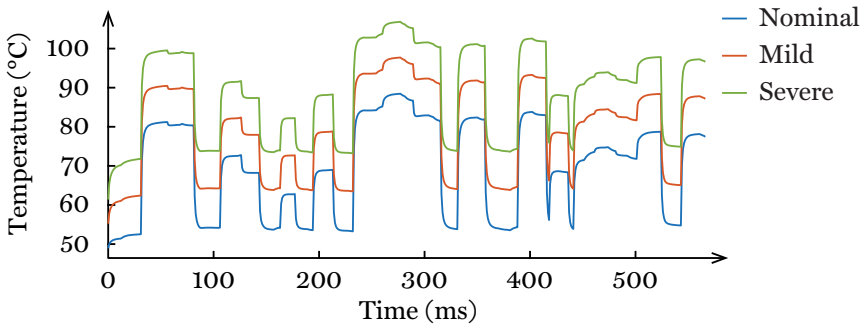

Figure 5.1: Example of temperature fluctuations due to process variation

the corresponding temperature profile. The result is depicted in Figure 5.1 by a blue line, which corresponds to the temperature of one of the processing elements; the experimental setup will be detailed in Section 5.8 and Section 5.9. Note that the temperature is always below 90°C. Let us now assume a mild deviation in the parameters from their nominal values in the direction that causes the leakage current to be higher, and let us perform temperature analysis once again. The result is the orange line in Figure 5.1; the maximum temperature is approaching 100°C. Finally, we repeat the experiment considering a severe deviation in the parameters in the same direction and observe the green line in Figure 5.1; in this case, the maximum temperature is almost 110°C.

Suppose now that the designer is tuning a solution constrained by a maximum temperature of 90°C, and that the designer is guided exclusively by the nominal values of the process parameters. In this scenario, even with mild deviations, the electrical circuits might be burnt. Another path that the designer might take is to design the system for severe conditions. This scenario, however, could easily lead to a system that is too conservative and overdesigned.

The conclusion drawn from the example given above is that the presence of uncertainty has to be adequately addressed in order to pursue efficiency and robustness. Nevertheless, the majority of the literature related to power, temperature, and reliability analysis of electronic systems ignores this important aspect; see, for instance, [86, 89, 91, 109, 121]. This negligence is also present in the analysis and optimization described in Chapter 3. Therefore, the goal of this chapter is to eliminate this concern in the case of process variation.

5.3 Problem Formulation

Assume the system model given in Section 2.1. Suppose that the system depends on a number of process parameters that are uncertain at the design stage. Once the fabrication process yields a particular outcome, the process parameters take certain values and stay unchanged thereafter. However, these values

are different for different fabricated chips, and they vary within each fabricated chip, since, in general, the variability due to process variation is not uniform. As emphasized earlier, this variability leads to such phenomena as deviations in power from the nominal values and, therefore, to deviations in temperature from the temperature corresponding to the nominal power consumption.

Each process parameter is a characteristic of a single transistor; consider, for instance, the effective channel length. Therefore, each device in the electrical circuit can potentially have a different value for this parameter. The process parameters can then be modeled as a stochastic process

$$u : \Omega \times \mathbb{R}^2 \to \mathbb{R}^{n_u}$$

that is defined on a suitable probability space $(\Omega, \mathcal{F}, \mathbb{P})$ (see Appendix A.2) and a two-dimensional plane and takes values in $\mathbb{R}^{n_u}$ where n_u is the number of the process parameters. For practical computations, the stochastic process is discretized, and each processing element is modeled via a finite set of random variables. The uncertain parameters of the problem are then defined as

$$\boldsymbol{u} = (u_i)_{i=1}^{n_u} : \Omega \to \mathbb{R}^{n_u}$$

where the union of all random variables of all processing elements is arranged into a single random vector, and n_u is redefined to be the number of elements that this vector has. Given the above setting, our goal is twofold as follows.

First, we are to develop a system-level framework for transient temperature (and hence power) analysis as well as dynamic steady-state analysis of electronic systems where power consumption and heat dissipation are stochastic due to their dependency on the parameters $\boldsymbol{u}$. The designer is required to specify (a) the probability distribution of $\boldsymbol{u}$ and (b) the dependency of the system's power consumption on $\boldsymbol{u}$, which can be given as a "black box." The framework should provide the designer with tools for analyzing the system under a given workload—without imposing constraints on the nature of this workload—and calculating the corresponding stochastic power $\mathbf{P}$ and stochastic temperature $\mathbf{Q}$ profiles with a desired level of accuracy and at low computational costs.

Second, taking the effect of process variation on power and temperature into consideration, we are to find the reliability function of the system and to develop a computationally efficient design-space-exploration scheme exploiting the proposed techniques for power, temperature, and reliability analysis.

5.4 Previous Work

As we elaborate in Section 1.2 and Section 1.5, uncertainty-unaware techniques are inadequate, and sampling methods—including Monte Carlo (MC) sampling—as a means of uncertainty quantification are computationally expensive. In order to overcome the limitations of deterministic approaches and,

at the same time, to eliminate or at least mitigate the computational costs associated with direct sampling, a number of probabilistic techniques have been introduced, which are discussed below. We are particularly interested in power and temperature and, therefore, pay special attention to these metrics in the exposition. Since the static component of power consumption is influenced by process variation the most, due mainly to the leakage current, the techniques discussed below focus primarily on the variability in this component.

A solely power-targeted but temperature-aware solution is proposed in [11] where the working force of the analysis is MC sampling with partially precomputed data. A learning-based approach is presented in [59] in order to estimate the maximum temperature under the static steady-state condition; recall Section 3.3. Temperature-related issues originating from process variation are also considered in [58] where a statistical model of the static steady-state temperature is derived based on the linearity of Gaussian distributions and time-invariant systems. A stochastic temperature simulator targeted at the static steady state is developed in [51] using the polynomial chaos (PC) decomposition and the continuous Karhunen–Loève (KL) decomposition; see Appendix A.4 and Appendix A.7. A stochastic collocation [68, 120] approach to static steady-state analysis is presented in [69], which relies on the KL decomposition and on Newton polynomials for interpolation. In [99], PC expansions are employed in order to estimate the static power of entire chips. The KL decomposition is utilized in [6] for calculating static power. Static power is also quantified in [7] via the PC and KL techniques. The same combination of tools is employed in [117] and [44] in order to analyze the response of interconnect networks and power grids, respectively, under process variation.

The last five of the above techniques, that is, [6, 7, 44, 99, 117], perform only probabilistic power analysis and ignore the interdependence between power and temperature described in Section 2.2. The other ones are temperature-related approaches, but none of them attempts to tackle probabilistic transient analysis, that is, to compute the probability distributions of power and temperature that evolve over time. However, such transient curves are of practical importance. First, certain procedures cannot be undertaken without knowledge of time-dependent variations; one example of this is reliability optimization concerned with thermal-cycling fatigue, which is discussed in Section 3.6. Second, the static steady-state assumption that is considered, for instance, in [51, 58, 59, 69] can rarely be justified, since power is not invariant in reality.

In addition, one frequently encounters the assumption that power and temperature follow *a priori* known probability distributions; Gaussian and log-normal distributions are popular choices; see, for instance, [6, 58, 103]. However, this assumption often fails in practice—which is also noted in [58] regarding the normality of the leakage current—due to (a) the nonlinear dependence of power on process parameters and (b) the nonlinear interdependence between power and temperature. In order to illustrate this, let us return to the example given in Section 5.2 and assume the widespread Gaussian model

of the effective channel length. We can then simulate the example 10^4 times and apply the Jarque–Bera test of normality to the collected temperature samples as well as to the samples obtained by passing the temperature samples through the log transformation. We observe that the null hypothesis, which avers that the data are from an unspecified Gaussian distribution, is firmly rejected in both cases at a significance level of 5% [90]. This means that, if the null hypothesis were true, the probability of observing these data would be less than 0.05. Consequently, the distribution is very unlikely to be Gaussian or log-normal, which can also be seen in Figure 5.6 shown in Section 5.9.

One can observe in the above discussion that the overwhelming majority of the literature related to temperature in the presence of process variation relies on static steady-state temperature analysis, which is inadequate in practice, and the other two types of temperature analysis—transient analysis and dynamic steady-state analysis—are not given enough attention. However, as discussed earlier, their availability is of practical importance to the designer.

Let us now discuss reliability analysis. Reliability analysis is probabilistic by nature. However, certain components of a reliability model can be treated as either stochastic or deterministic, depending on the phenomena that the model is designed for. Temperature is an example of such a component: it can be considered deterministic if the effect of process variation on temperature is neglected, and it can be considered stochastic if this effect is accounted for. The former scenario is the one that is typically addressed in the literature related to reliability. For instance, the reliability model proposed in [118] has a treatment of process variation; however, temperature is included in the model as a deterministic quantity. In [27], a design methodology that minimizes energy consumption of and temperature-induced wear on multiprocessor systems is introduced; yet neither energy nor temperature is modeled with an awareness of uncertainty due to process variation. A similar observation can be made with respect to the work reported in [28] where a reinforcement-learning algorithm is used in order to improve the lifetime of multiprocessor systems. An extensive survey of reliability-aware system-level design techniques given in [26] confirms the trend emphasized above: the widespread device-level models of failure mechanisms generally ignore the impact of process variation on temperature. However, it is unwise to assume that temperature is deterministic, since it could lead to a substantial yield loss.

An example of a different kind is the work presented in [69]. It introduces a statistical simulator for reliability analysis under process variation and does consider temperature as a stochastic parameter. However, as discussed previously, this study is limited to static steady-state temperatures. Moreover, the reliability analysis that it presents is an analysis of maximum temperatures without any direct connection to the common failure mechanisms [37]. The work in [62] is worth mentioning as well. Although it is not directly concerned with reliability analysis, it considers aging variation together with process variation and presents a framework for timing analysis based on MC sampling.

To summarize, the prior techniques for probabilistic power and temperature analysis are restricted in use due to one or several of the following traits: (a) based on MC simulations (potentially slow) [11], (b) limited to power analysis [6, 7, 11, 44, 99, 117], (c) ignoring the power-temperature interplay [6, 7, 44, 51, 99, 117], (d) limited to the static steady-state temperature [51, 58, 59, 69], (e) exclusive focus on the maximum temperature [59], and (f) *a priori* chosen distributions of power and temperature [6, 58, 103]. The designer's toolbox does not yet include tools for transient analysis and dynamic steady-state analysis under process variation, which are of great importance for certain applications. Furthermore, reliability models lack a flexible approach to taking the effect of process variation on power and temperature into consideration.

5.5 Proposed Solution

Due to their inherent complexity, uncertainty-quantification problems are often viewed as approximation problems: one constructs a computationally efficient surrogate for the system under consideration and then studies this representation instead of the original system. In this chapter, we resort to the PC decomposition, which is thoroughly introduced in Appendix A.7, for the construction of such a lightweight surrogate for the quantity of interest g. The technique decomposes stochastic quantities into infinite series of mutually orthogonal polynomials operating on random variables. These series are an attractive alternative to MC sampling, since they possess much faster convergence properties and provide succinct and intuitive representations of the system's responses to stochastic inputs. Having obtained an adequate polynomial surrogate for g, we utilize it for calculating the desired statistics about g, such as its cumulative distribution function (CDF), probability density function (PDF), probabilistic moments, and probabilities of various events.

The solution is consolidated in a framework for analyzing electronic systems that are subject to uncertainty due to process variation. The framework is flexible in the way that it models diverse probability distributions of the uncertain parameters specified by the designer. Examples of such parameters include the effective channel length and gate oxide thickness. Moreover, there are no assumptions regarding the probability distribution of the quantity of interest, as this distribution is unlikely to be known *a priori*. Examples of such quantities include power and temperature profiles. The proposed technique is capable of capturing arbitrary joint effects of the uncertain parameters on the system, since the impact of these parameters is introduced into the framework as a "black box," which is also defined by the designer. In particular, it allows the power-temperature interplay to be taken into account with no effort.

Leveraging the proposed framework, we extend the deterministic transient analysis and the deterministic dynamic steady-state analysis presented in Section 3.2 and Section 3.4, respectively, by taking account of process variation.

Moreover, the framework allows us to enrich the reliability analysis presented in Section 2.4, which is based on state-of-the-art reliability models, by taking the effect of process variation on temperature into consideration.

We illustrate the proposed framework by considering two important process parameters that are affected by process variation, namely the effective channel length and gate oxide thickness; note, however, that our approach can be applied to other parameters as well. Furthermore, we utilize the framework in order to construct a computationally efficient design-space-exploration procedure targeted at minimizing energy consumption under a set of probabilistic constraints on the temperature and lifetime of the system at hand.

In the following sections, we present our general approach to uncertainty analysis and then specialize it for the aforementioned scenarios.

5.6 Uncertainty Analysis

The key building block of the solutions that we develop in the subsequent sections is the uncertainty-quantification technique presented in this section. The task of this technique is to propagate uncertainty through the system, from a set of inputs to a set of outputs. The inputs are the uncertain parameters $\boldsymbol{u}$, and the outputs are the quantities that the designer is interested in studying. The former could be, for instance, the effective channel length, gate oxide thickness, and threshold voltage, and the latter could be, for instance, the temperature profile, energy consumption, and maximum temperature of the system.

The major stages of our general approach are depicted in Figure 5.2. At Stage 1, the quantity of interest g and the uncertain parameters $\boldsymbol{u}$ are specified. The quantity is given as a "black-box" function that operates on a particular outcome of the parameters. In order to evaluate g, the designer is implicitly required to specify the system model being considered, which also includes a power model and a temperature model. At Stage 2, the uncertain parameters $\boldsymbol{u}$ are to be transformed into independent random variables $\boldsymbol{z}$, since independence is a prerequisite of the subsequent calculations. At Stage 3, a surrogate for g is constructed by means of the PC decomposition. At Stage 4, the computed expansion is processed in order to obtain the desired statistics about g.

5.6.1 Problem Formulation

The problem formulation is problem specific. Thus, in this general section, we operate on unspecified g and $\boldsymbol{u}$; more concrete discussions about Stage 1 are postponed to later sections where we start to apply the proposed methodology to particular problems. In addition, for convenience, g is assumed to take values in $\mathbb{R}$ here, which is generalized to $\mathbb{R}^n$ later on. Lastly, there is one recurring aspect about $\boldsymbol{u}$ that is worth discussing before going any further.

A description of $\boldsymbol{u}$ that is supplied by the designer is an input to our probabilistic analysis. A proper way to describe a set of random variables is to specify

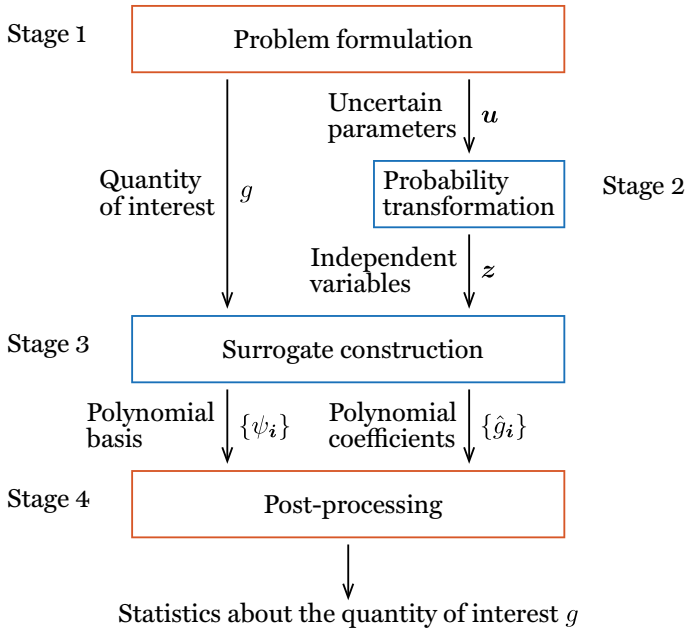

Figure 5.2: Overview of the proposed solution for analyzing electronic systems under process variation

their joint distribution function; see Appendix A.2. In practice, however, such exhaustive information is often unavailable, which is due mainly to the high dimensionality and intricate dependencies inherent in real-life problems.

A more realistic assumption is the knowledge of the marginal distributions $\{F_i\}_{i=1}^{n_u}$ and correlation matrix $\mathbb{C}\mathrm{orr}(\boldsymbol{u})$ of $\boldsymbol{u}$. However, these are not sufficient to reconstruct the joint distribution of $\boldsymbol{u}$ in general. Nevertheless, it can be approximated well by accompanying the available marginals with a carefully constructed copula [79], which makes $\boldsymbol{u}$ fully specified; see Appendix A.2. Specifically, one constructs a Gaussian copula based on $\{F_i\}_{i=1}^{n_u}$ and $\mathbb{C}\mathrm{orr}(\boldsymbol{u})$ as a part of the Nataf transformation [74], which is introduced in Appendix A.4. This Gaussian copula is defined in terms of an auxiliary correlation matrix.

Without loss of generality, we target the above practical scenario in our experiments in this section and the next one. However, it should be understood that, even though the marginal distributions and the above copula prescribe a joint distribution for $\boldsymbol{u}$, this distribution is an approximation rather than the true one. If the joint distribution is available, it should be used instead.

5.6.2 Probability Transformation

Mutual independence of random variables is required by PC expansions. In general, however, the individual variables in $\boldsymbol{u} : \Omega \to \mathbb{R}^{n_u}$ are dependent. Therefore, our foremost task is to transform $\boldsymbol{u}$ into a vector with independent components in order to fulfill the requirement; see Stage 2 in Figure 5.2. To this end, an adequate transformation should be performed, depending on the available information [36]. Denote such a transformation by

$$\boldsymbol{u} = \mathbb{T}(\boldsymbol{z}) \tag{5.1}$$

where $\boldsymbol{z} : \Omega \to \mathbb{R}^{n_z}$ is a random vector with n_z independent components, and $\mathbb{T} : \mathbb{R}^{n_z} \to \mathbb{R}^{n_u}$. The quantity of interest g can now be computed as

$$g(\boldsymbol{u}) = (g \circ \mathbb{T})(\boldsymbol{z}) = g(\mathbb{T}(\boldsymbol{z})).$$

Correlated random variables can be transformed into linearly uncorrelated ones via the KL decomposition described in Appendix A.4 and shown in Equation A.6. In addition, if the correlated variables form a Gaussian vector, the uncorrelated variables are mutually independent. In the general case (non-Gaussian), the most prominent solutions for attaining independence are the Rosenblatt transformation [95] and the Nataf transformation mentioned earlier. Rosenblatt's approach is suitable when the joint distribution of $\boldsymbol{u}$ is known. However, as emphasized in Section 5.6.1, such information is rarely available. Marginal distributions and a correlation matrix are more likely to be given, which are sufficient for the Nataf transformation; see Appendix A.4.

Apart from the extraction of the independent variables $\boldsymbol{z}$, an essential operation at this stage is model order reduction, since the number of stochastic dimensions—that is, the dimensionality of $\boldsymbol{z}$—directly impacts the complexity of the rest of the computations. This operation is frequently treated as a part of the KL decomposition, and it is also elaborated on in Appendix A.4.

5.6.3 Surrogate Construction

In order to obtain a computationally efficient and convenient characterization of g, we utilize the PC decomposition with nonintrusive spectral projections [120]. The corresponding mathematical foundation is given in Appendix A.7.

Assume that g as a function of $\boldsymbol{z}$ belongs to $L^2(\Omega, \mathcal{F}, \mathbb{P})$; see Appendix A.2. Then g is expanded into the following series at Stage 3 in Figure 5.2:

$$g \approx \mathcal{C}_{l_c}^{n_z}(g) = \sum_{\boldsymbol{i} \in \mathcal{I}_{l_c}^{n_z}} \hat{g}_{\boldsymbol{i}} \psi_{\boldsymbol{i}} \tag{5.2}$$

where $l_c \in \mathbb{N}_0$ is the level of the expansion, $\boldsymbol{i} = (i_k) \in \mathbb{N}_0^{n_z}$ is an index, $\mathcal{I}_{l_c}^{n_z}$ is an index set, and $\{\psi_{\boldsymbol{i}} : \boldsymbol{i} \in \mathcal{I}_{l_c}^{n_z}\}$ are orthonormal polynomials in n_z variables whose orders are specified by the corresponding elements of $\boldsymbol{i}$.

It is clear that the first step toward a polynomial expansion is the choice of a suitable polynomial basis, which is typically made based on the Askey scheme of orthogonal polynomials [120]. This step is crucial, as the rate of convergence of PC expansions depends on it. Although there are no rules that guarantee the optimal choice [65], there are best practices suggesting that one should be guided by the probability distributions of the random variables that drive the stochastic system at hand. For instance, when a random variable follows a beta distribution, it is worth trying the Jacobi basis first; on the other hand, the Hermite basis is preferable for Gaussian distributions.

As shown in Equation A.25 and Equation A.27, each coefficient $\hat{g}_{\boldsymbol{i}}$ is an n_z-dimensional integral of the product of g and $\psi_{\boldsymbol{i}}$. In general, this integral should be computed numerically as described in Appendix A.5. Specifically, an adequate n_z-dimensional quadrature $\mathcal{Q}_{l_q}^{n_z}$, which is a set of n_z-dimensional points accompanied by a set of scalar weights, is utilized. The result is

$$\hat{g}_{\boldsymbol{i}} \approx \mathcal{Q}_{l_q}^{n_z}(g\psi_{\boldsymbol{i}}) = \sum_{j \in \mathcal{J}_{l_q}^{n_z}} (g \circ \mathbb{T})(\boldsymbol{z}_j)\psi_{\boldsymbol{i}}(\boldsymbol{z}_j)w_j \tag{5.3}$$

where $l_q \in \mathbb{N}_0$ is the quadrature's level, and $\{\boldsymbol{z}_j\} \subset \mathbb{R}^{n_z}$ and $\{w_j\} \subset \mathbb{R}$ are the corresponding points and weights, respectively, indexed by $\mathcal{J}_{l_q}^{n_z} \subset \mathbb{N}_0$. The operator $\mathcal{Q}_{l_q}^{n_z}$ is constructed via the Smolyak algorithm [101] as shown in Equation A.11. An important aspect to note about this construction is that it is a combination of a number of cherry-picked operators identified by a certain index set denoted by $\mathcal{I}_{l_q}^{n_z} \subset \mathbb{N}_0^n$. Let us discuss the content of $\mathcal{I}_{l_c}^{n_z}$ and $\mathcal{I}_{l_q}^{n_z}$.

The standard choice of $\mathcal{I}_{l_c}^{n_z}$ in Equation 5.2 is the isotropic total-order index set, which can be seen in Equation A.12. *Isotropic* refers to constraining all dimensions identically, and *total-order* to the criterion used for constraining each dimension. Since $\psi_{\boldsymbol{i}}$ is a polynomial of total order at most l_c, and g is approximated by such a polynomial, the integrand in Equation 5.3 can be assumed to be a polynomial of total order at least $2l_c$. With this in mind, one typically constructs such a quadrature that is exact for polynomials of total order up to at least $2l_c$ [36]. More generally, the index set $\mathcal{I}_{l_c}^{n_z}$, which is used in Equation 5.2, and the index set $\mathcal{I}_{l_q}^{n_z}$, which implicitly determines the content of the index set $\mathcal{J}_{l_q}^{n_z}$ used in Equation 5.3, should be related as $\mathcal{I}_{l_c}^{n_z} \subseteq \mathcal{I}_{l_q}^{n_z}$.

In the case of Gaussian quadratures, which is a broad and potent class of quadratures introduced in Appendix A.5, a quadrature of level l_q is exact for polynomials of total order up to $2l_q + 1$ [47]. Therefore, in this very common case, an adequate quadrature can be constructed by ensuring that $l_q \geq l_c$.

An important generalization of the isotropic Smolyak algorithm in Equation A.11 is the anisotropic Smolyak algorithm [82]. The difference between the isotropic and anisotropic versions lies in the content of $\mathcal{I}_{l_q}^{n_z}$. In particular, the anisotropic total-order index set is defined as follows:

$$\mathcal{I}_{l_q}^{n_z} = \left\{ \boldsymbol{i} : \boldsymbol{i} \in \mathbb{N}_0^{n_z}, \langle \boldsymbol{c}, \boldsymbol{i} \rangle \leq l_q \min_{i=1}^{n_z} c_i \right\} \tag{5.4}$$

Algorithm 5.1: Construction of a polynomial chaos expansion

Input: Algorithm G // a subroutine evaluating $g \circ \mathbb{T}$
Output: $\hat{\boldsymbol{g}} \in \mathbb{R}^{n_c}$
1: **for** $i \leftarrow 1$ **to** n_q **do** // for each quadrature point $\boldsymbol{z}_i$
2: $\quad \boldsymbol{g}(i) \leftarrow$ Algorithm G($\boldsymbol{z}_i$)
3: **end for**
4: $\hat{\boldsymbol{g}} \leftarrow \Pi \boldsymbol{g}$
5: **return** $\hat{\boldsymbol{g}}$

where $\boldsymbol{c} = (c_i) \in \mathbb{R}^{n_z}$ with $c_i \geq 0$ for $i = 1, \ldots, n_z$ is a vector assigning importance weights to the dimensions, and $\langle \cdot, \cdot \rangle$ is the standard inner product in $\mathbb{R}^{n_z}$. Equation 5.4 plugged into Equation A.11 results in a sparse grid that is exact for the polynomial subspace that is obtained using the same index set.

The above approach allows one to exploit anisotropic behaviors that are present in many practical problems [82]. It provides fine control over the computational cost associated with the construction of PC expansions: a carefully chosen importance vector $\boldsymbol{c}$ in Equation 5.4 can significantly reduce the number of polynomial terms in Equation 5.2 and the number of quadrature points in Equation 5.3, which are needed for calculating the coefficients of those polynomial terms. The question, then, is in the choice of $\boldsymbol{c}$. When the KL decomposition is utilized as a part of $\mathbb{T}$ in Equation 5.1, a viable option in this regard is to rely on the variance contributions of the dimensions given by $\{\lambda_i\}_{i=1}^{n_u}$ in Equation A.6. Specifically, we let

$$c_i = \left(\frac{\lambda_i}{\sum_{j=1}^{n_u} \lambda_j} \right)^{\gamma} \tag{5.5}$$

for $i = 1, \ldots, n_z$ where $\gamma \in [0, 1]$ is a tuning parameter. The isotropic scenario can be recovered by setting $\gamma = 0$; other values of γ correspond to various levels of anisotropy with the maximum attained by setting $\gamma = 1$.

Once $\boldsymbol{z}$ has been identified, and l_c, l_q, and $\boldsymbol{c}$ have been chosen, the corresponding polynomial basis and quadrature stay the same for all quantities that one might be interested in studying. This observation is very important, as a lot of preparatory work can and should be done only once and then reused as needed. In particular, the construction in Equation 5.2 can be reduced to one matrix multiplication with a precomputed matrix, which we show next.

Let $n_c = \#\mathcal{I}_{l_c}^{n_z}$ be the cardinality of $\mathcal{I}_{l_c}^{n_z}$, which is the number of polynomial terms and coefficients in Equation 5.2. Let also $n_q = \#\mathcal{J}_{l_q}^{n_z}$ be the cardinality of $\mathcal{J}_{l_q}^{n_z}$, which is the number of quadrature points and weights in Equation 5.3. Furthermore, assume that the index sets $\mathcal{I}_{l_c}^{n_z}$ and $\mathcal{J}_{l_q}^{n_z}$ are given certain orderings so that one is able to refer to their elements using one-dimensional indices $i = 1, \ldots, n_c$ and $j = 1, \ldots, n_q$, respectively. Now, let

$$\Pi = (\psi_i(\boldsymbol{z}_j) w_j)_{i=1,j=1}^{i=n_c,j=n_q}. \tag{5.6}$$

The element of Π on row i and column j is polynomial i evaluated at quadrature point j and multiplied by quadrature weight j. The matrix Π is referred to as the projection matrix. Using Π, the coefficients $\{\hat{g}_{\boldsymbol{i}} : \boldsymbol{i} \in \mathcal{I}_{l_c}^{n_z}\}$ in Equation 5.2 can now be trivially computed as follows:

$$\hat{\boldsymbol{g}} = \Pi \boldsymbol{g} \tag{5.7}$$

where

$$\hat{\boldsymbol{g}} = (\hat{g}_i)_{i=1}^{n_c} \text{ and}$$
$$\boldsymbol{g} = ((g \circ \mathbb{T})(\boldsymbol{z}_i))_{i=1}^{n_q}.$$

It can be seen that this formula is a matrix version of Equation 5.3. The matrix Π is the one that should be precomputed and stored for future use.

The pseudocode for a procedure that computes PC expansions by leveraging the projection matrix Π is given in Algorithm 5.1 where Algorithm G stands for a subroutine that calculates $g \circ \mathbb{T}$ for a given $\boldsymbol{z}$, which is problem specific.

Let us summarize this subsection. In order to give a probabilistic characterization of the quantity of interest g, we construct a polynomial expansion of this quantity as shown in Equation 5.2. The coefficients of this expansion are found by means of a suitable multivariate quadrature as shown in Equation 5.3. The quadrature is constructed via the Smolyak formula given in Equation A.11. The index set used in both Equation 5.2 and Equation A.11 is the one given in Equation 5.4 where the anisotropic weights are set according to Equation 5.5. For computational efficiency, the projection matrix defined in Equation 5.6 is to be calculated, stored, and used as illustrated in Algorithm 5.1.

5.6.4 Post-Processing

Due to the properties of the PC decomposition—in particular, the orthogonality of the basis discussed in Appendix A.7—the obtained polynomial representation in Equation 5.2 allows various statistics about g to be estimated with little effort, which is the subject of Stage 4 in Figure 5.2. The reason that this estimation is straightforward is that the function given in Equation 5.2 is nothing more than a polynomial; hence, it is easy to interpret and easy to evaluate.

Let us find, for example, the expectation and variance of g. Since the first polynomial $\psi_{\mathbf{0}}$ in a normalized polynomial basis is unity by definition [120],

$$\mathbb{E}\psi_{\mathbf{0}} = 1.$$

Hence, using the orthogonality property in Equation A.26, we conclude that

$$\mathbb{E}\psi_{\boldsymbol{i}} = 0$$

for $\boldsymbol{i} \in \mathcal{I}_{l_c}^{n_z} \setminus \{\mathbf{0}\}$. Consequently, the expected value and variance of g have the following straightforward expressions:

$$\begin{aligned} &\mathbb{E}g = \hat{g}_{\mathbf{0}} \text{ and} \\ &\mathbb{V}\mathrm{ar}\,(g) = \sum_{\boldsymbol{i} \in \mathcal{I}_{l_c}^{n_z} \setminus \{\mathbf{0}\}} \hat{g}_{\boldsymbol{i}}^2, \end{aligned} \tag{5.8}$$

respectively. It can be seen that the PC decomposition provides analytical formulae, based solely on the coefficients, for these probabilistic moments.

The CDF and PDF of g can be estimated by a sampling method applied to Equation 5.2, which is typically followed by kernel density estimation [46] or a similar technique. The sampling in this context can be better understood by rewriting Equation 5.2, which is given in terms of operators, as follows:

$$(g \circ \mathbb{T})(\boldsymbol{z}) \approx \mathcal{C}_{l_c}^{n_z}\,(g)\,(\boldsymbol{z}) = \sum_{\boldsymbol{i} \in \mathcal{I}_{l_c}^{n_z}} \hat{g}_{\boldsymbol{i}} \psi_{\boldsymbol{i}}(\boldsymbol{z}).$$

Here the aforementioned operators are applied to an outcome of $\boldsymbol{z}$ drawn from the corresponding distribution. Consequently, each sample is a trivial evaluation of a polynomial; therefore, sampling methods are computationally cheap in this case. Furthermore, probabilities of various events can be estimated in a similar way, and global and local sensitivity analysis of deterministic and stochastic quantities can be readily conducted on the expansion.

Remark 5.1. *The development given in this section remains valid even when g is a multidimensional quantity from the standpoint of the number of outputs. In this case, it is convenient to consider g as a row vector with an appropriate number of elements. All the operations that involve g—such as those given in Equation 5.2, Equation 5.3, and Equation 5.8—should then be performed elementwise. In Equation 5.7 and Algorithm 5.1, $\mathbf{g}$ and $\hat{\mathbf{g}}$ are treated as matrices with n_c rows, and g_i as a row vector. The output of Algorithm G in Algorithm 5.1 is assumed to be automatically reshaped into a row vector.*

In the following, we apply the uncertainty analysis presented here to a number of concrete problems, namely transient and dynamic steady-state power and temperature analysis as well as reliability analysis and optimization.

5.7 Transient Analysis

In this section, we develop a technique for probabilistic transient power and temperature analysis of electronic systems using the uncertainty-unaware approach presented in Section 3.2.2 combined with the machinery described in Section 5.6. Our goal is to obtain a technique that preserves the gradual solution process that is at the heart of transient analysis; see Section 3.2. In this way, the designer retains fine control over transient calculations.

5.7.1 Problem Formulation

As stated in Section 5.6, the designer is supposed to decide on the system model that produces the quantity of interest. Assuming the general system and temperature models given in Section 2.1 and Section 2.3, respectively, the only component that is left to define is the power model, since it is what introduces the actual workload into the system. Denote the power model by

$$\boldsymbol{p} = f(i, \boldsymbol{q}, \boldsymbol{u}) \tag{5.9}$$

where $f : \mathbb{N}_+ \times \mathbb{R}^{n_p} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_p}$ is a function that evaluates the power consumption $\boldsymbol{p} \in \mathbb{R}^{n_p}$ of the processing elements at time step i given their heat dissipation $\boldsymbol{q} \in \mathbb{R}^{n_p}$ and an assignment of the uncertain parameters $\boldsymbol{u} \in \mathbb{R}^{n_u}$.

Remark 5.2. *It should be understood that $\boldsymbol{p}$, $\boldsymbol{q}$, and $\boldsymbol{u}$ are random vectors in general, and that f consumes $\boldsymbol{q}(\omega)$ and $\boldsymbol{u}(\omega)$ and yields $\boldsymbol{p}(\omega)$ for some particular outcome $\omega \in \Omega$. The function f* per se *is purely deterministic.*

The designer can choose any f. For instance, it can be a closed-form formula or a piece of code. The only assumption we make is that f is smooth in $\boldsymbol{z}$ and, when viewed as a random variable, belongs to $L^2(\Omega, \mathcal{F}, \mathbb{P})$ (see Appendix A.2), which is generally applicable to most physical systems [120]. The definition of f is flexible enough to account for such phenomena as the interdependence between power and temperature discussed in Section 2.2.

Our solution to transient analysis under process variation is based on the one presented in Section 3.2.2. The major difference is that Equation 3.5 implicitly operates on stochastic quantities in the context of this chapter. Consequently, the recurrent solution in Equation 3.6, that is,

$$\boldsymbol{s}_i = \mathbf{E}\boldsymbol{s}_{i-1} + \mathbf{F}\boldsymbol{p}_i \tag{5.10}$$

for $i = 1, \ldots, n_s$, is stochastic as well. In the deterministic case, it can be readily employed in order to perform transient power and temperature analysis. In the probabilistic case, however, the situation is substantially different, since $\boldsymbol{p}_i$ and hence $\boldsymbol{s}_i$ and $\boldsymbol{q}_i$ are stochastic quantities. Moreover, at each step, $\boldsymbol{p}_i$ is an arbitrary transformation of the uncertain parameters $\boldsymbol{u}$ and stochastic temperature $\boldsymbol{q}_i$, which results in a random vector with a generally unknown probability distribution. Furthermore, $\boldsymbol{p}_i$, $\boldsymbol{q}_i$, $\boldsymbol{s}_i$, and $\boldsymbol{u}$ are dependent random vectors, since the first three are functions of the last one. Therefore, the operations in Equation 5.10 are to be performed on dependent random vectors with arbitrary distributions, which, in general, have no closed-form solutions.

Let us now summarize Stage 1 in Figure 5.2. In this section, the quantity of interest g is the transient power and temperature profiles—denoted by $\mathbf{P}$ and $\mathbf{Q}$, respectively—that correspond to the workload specified by f as shown in Equation 5.9. Regarding the uncertain parameters $\boldsymbol{u}$, they are left unspecified, since the construction below is general with respect to $\boldsymbol{u}$; however, a concrete

scenario will be considered in the next section, Section 5.8. We now proceed directly to Stage 3, as Stage 2 requires no additional attention in this case.

5.7.2 Surrogate Construction

The goal now is to transform the recurrence in Equation 5.10 in such a way that the distributions of power and temperature can be efficiently estimated. Based on the methodology presented in Section 5.6, we construct a PC expansion of f so that it can then be propagated through the recurrence in Equation 5.10 in order to obtain a PC expansion of power and a PC expansion of temperature.

The expansion of $\boldsymbol{p}_i$, which is required in Equation 5.10 and computed via f shown in Equation 5.9, is as follows:

$$\mathcal{C}_{l_c}^{n_z}\left(\boldsymbol{p}_i\right) = \sum_{\boldsymbol{j} \in \mathcal{I}_{l_c}^{n_z}} \hat{\boldsymbol{p}}_{i\boldsymbol{j}} \psi_{\boldsymbol{j}}$$

where $\{\psi_{\boldsymbol{j}} : \mathbb{R}^{n_z} \to \mathbb{R}\}$ are the basis polynomials, $\{\hat{\boldsymbol{p}}_{i\boldsymbol{j}}\} \subset \mathbb{R}^{n_p}$ are the corresponding coefficients, and the index set $\mathcal{I}_{l_c}^{n_z}$ is the one given in Equation 5.4. In addition, Remark 5.1 is worth recalling. It can be seen in Equation 5.10 that, due to the linearity of the operations involved in the recurrence, $\boldsymbol{s}_i$ attains such a PC expansion that has the same structure as the expansion of $\boldsymbol{p}_i$. The recurrence in Equation 5.10 can then be rewritten as follows:

$$\mathcal{C}_{l_c}^{n_z}\left(\boldsymbol{s}_i\right) = \mathbf{E}\,\mathcal{C}_{l_c}^{n_z}\left(\boldsymbol{s}_{i-1}\right) + \mathbf{F}\,\mathcal{C}_{l_c}^{n_z}\left(\boldsymbol{p}_i\right)$$

for $i = 1, \ldots, n_s$. Consequently, there are two interwoven PC expansions: one is for power, and the other for temperature. The two expansions have the same polynomial basis but different coefficients. In order to understand the structure of the above formula better, let us spell it out as

$$\sum_{\boldsymbol{j} \in \mathcal{I}_{l_c}^{n_z}} \hat{\boldsymbol{s}}_{i\boldsymbol{j}} \psi_{\boldsymbol{j}} = \sum_{\boldsymbol{j} \in \mathcal{I}_{l_c}^{n_z}} \left(\mathbf{E}\hat{\boldsymbol{s}}_{i-1,\boldsymbol{j}} + \mathbf{F}\hat{\boldsymbol{p}}_{i\boldsymbol{j}}\right) \psi_{\boldsymbol{j}}.$$

Making use of the orthogonality property, which can be seen in Equation A.26, we obtain the following recurrence:

$$\hat{\boldsymbol{s}}_{i\boldsymbol{j}} = \mathbf{E}\hat{\boldsymbol{s}}_{i-1,\boldsymbol{j}} + \mathbf{F}\hat{\boldsymbol{p}}_{i\boldsymbol{j}} \tag{5.11}$$

for $i = 1, \ldots, n_s$ and $\boldsymbol{j} \in \mathcal{I}_{l_c}^{n_z}$. The coefficients $\{\hat{\boldsymbol{p}}_{i\boldsymbol{j}}\}$ needed in this recurrence are found using a suitable quadrature $\mathcal{Q}_{l_q}^{n_z}$ (see Appendix A.5) as follows:

$$\hat{\boldsymbol{p}}_{i\boldsymbol{j}} = \mathcal{Q}_{l_q}^{n_z}\left(\boldsymbol{p}_i \psi_{\boldsymbol{j}}\right)$$

for $i = 1, \ldots, n_s$ and $\boldsymbol{j} \in \mathcal{I}_{l_c}^{n_z}$ where the operation is elementwise, and it can be efficiently performed by virtue of the projection matrix in Equation 5.6.

The final step of the construction process is to combine Equation 5.11 with Equation 3.5b in order to obtain the coefficients of the PC expansion of the

temperature vector $\boldsymbol{q}_i$. Note that, since power depends on temperature, which is discussed in Section 2.2, at each step of the recurrence in Equation 5.11, the computation of $\hat{\boldsymbol{p}}_{i\boldsymbol{j}}$ via f should be done with respect to the expansion of $\boldsymbol{q}_{i-1}$.

The construction process of the stochastic power and temperature profiles is estimated to have the following time complexity per time step:

$$\mathcal{O}\left(n_n^2 n_c + n_n n_p n_q n_c + n_q f(n_p)\right)$$

where n_n, n_p, n_c, and n_q are the number of thermal nodes, processing elements, polynomial terms, and quadrature points, respectively; and $f(n_p)$ denotes the contribution of the power model shown in Equation 5.9. The expression can be detailed further by expanding n_c and n_q. Assuming the isotropic total-order index set in Equation A.12, n_c can be calculated as shown in Equation A.13. This formula behaves as ${n_z}^{l_c}/l_c!$ in the limit with respect to n_z. Regarding n_q, for quadratures based on the full tensor product given in Equation A.9, $\log(n_q) \propto n_z$, which means that the dependency of n_q on n_z is exponential.

It can be seen that the theory of PC expansions suffers from the curse of dimensionality [36, 120]: when n_z increases, the number of polynomial terms as well as the complexity of the corresponding coefficients exhibit growth, which is exponential without special treatments. The problem does not have a general solution and is one of the central topics of many ongoing studies.

We mitigate the curse of dimensionality by (a) keeping the number of stochastic dimensions low via model order reduction, which is a part of $\mathbb{T}$ shown in Equation 5.1 and is based on the KL decomposition described in Appendix A.4, and (b) utilizing efficient construction techniques, which is discussed in Section 5.6.3 and Appendix A.5. For instance, in the case of isotropic integration grids based on the Smolyak algorithm and Gaussian quadratures, $\log(n_q) \propto \log(n_z)$, which means that the dependency of n_q on n_z is only polynomial [47]. Anisotropic constructions allow for further reduction.

Let us summarize Stage 3 in Figure 5.2. Recall the stochastic recurrence in Equation 5.10 where, in the presence of dependencies, an arbitrary function $\boldsymbol{p}_i$ (since it is based on f as shown Equation 5.9) of the stochastic temperature $\boldsymbol{q}_i$ and the uncertain parameters $\boldsymbol{u}$ needs to be evaluated and combined with the random vector $\boldsymbol{s}_i$. This recurrence has been replaced with a purely deterministic one in Equation 5.11. More generally, the whole system, including the temperature model in Equation 2.5, has been substituted with a lightweight surrogate defined by a set of polynomials $\{\psi_{\boldsymbol{j}}\}$, a set of coefficients $\{\hat{\boldsymbol{p}}_{i\boldsymbol{j}}\}$ for power, and a set of coefficient $\{\hat{\boldsymbol{q}}_{i\boldsymbol{j}}\}$ for temperature. These quantities constitute the desired stochastic power and temperature profiles, and these profiles are ready to be analyzed at each step of the process as described in Section 5.6.4.

Before we proceed, let us draw attention to the ease and generality of taking process variation into consideration using the proposed approach. The description given above is delivered from any explicit formula of any particular process parameter. In contrast, the solutions from the literature related to

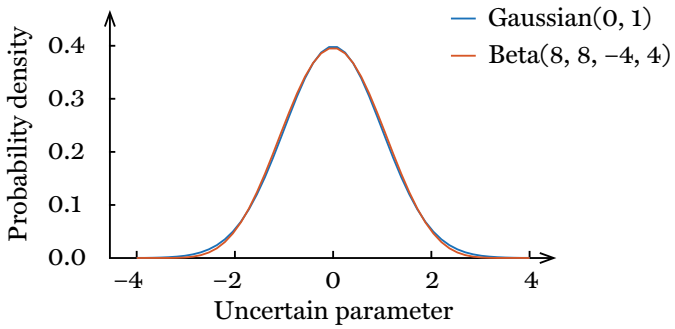

Figure 5.3: Beta distribution fitted to the standard Gaussian distribution

process variation are typically based on ad hoc expressions and should be individually tailored by the designer to each new parameter; see, for instance, [7, 44, 51]. The proposed framework provides great flexibility in this regard.

5.8 Transient Analysis: Illustrative Application

In this section, we consider a particular application of the proposed approach to probabilistic transient analysis presented in Section 5.7 in order to illustrate its usage in practice. We begin by describing the scenario being considered.

5.8.1 Problem Formulation

At Stage 1 in Figure 5.2, the quantity of interest g is the transient power and temperature profiles corresponding to a given workload. Let us now specify the parameters $\boldsymbol{u}$ that make g uncertain to the designer of the system.

As discussed in Section 2.2, the total dissipation of power is composed of two major components: dynamic and static. The influence of process variation on dynamic power is known to be negligibly small [103]; on the other hand, the variability in static power is substantial, with the subthreshold leakage current contributing the most [58, 59]. With this in mind, we focus our attention on the subthreshold leakage and, more specifically, on the effective channel length, which is denoted by u, since it has the strongest influence on this leakage (including its impact on other important parameters such as the threshold voltage) and is severely deteriorated by process variation [12].

It is well known that the dispersion of the effective channel length around its nominal value resembles the bell shape of Gaussian distributions. Therefore, such variations are often conveniently modeled using Gaussian random variables [6, 11, 44, 51, 58, 59, 69, 99, 103]. Due to the underlying physics and for demonstration purposes, we take a step further and embed into the model the fact that the effective channel length—occupying the space between the

drain and source of a transistor—cannot be arbitrarily large or take negative values, which Gaussian distributions allow it to do. In other words, we require the model of u to have a bounded support. To this end, we propose to model the effective channel length and other physically bounded parameters using the four-parameter family of beta distributions as

$$u \sim \mathrm{Beta}(a, b, c, d) \tag{5.12}$$

where a and b are the shape parameters, and c and d are the left and right bounds of the support, respectively. The parameters a and b can be chosen so that the frequently observed bell shape is preserved. An illustration is given in Figure 5.3 where a beta distribution is fitted to the standard Gaussian distribution; alternatively, one can match probabilistic moments. It can be seen that the curves are nearly indistinguishable; however, the beta one has a bounded support $[-4, 4]$, which can potentially lead to more realistic models.

The variability in u is split into global and local parts [11, 58, 99], which are denoted by u_{glob} and u_{loc}, respectively. The former can be treated as a composition of inter-lot, inter-wafer, and inter-die variations, and the latter as a composition of intra-die variations. The variability u_{glob} is assumed to be shared by all the n_p processing elements, whereas each processing element is assumed to have its own local parameter $u_{\mathrm{loc},i}$. The effective channel length of processing element i is then modeled using the following formula:

$$u_i = u_{\mathrm{nom}} + u_{\mathrm{glob}} + u_{\mathrm{loc},i}$$

where u_{nom} is the nominal value of the effective channel length. Consequently, the uncertain parameters of the problem are

$$\boldsymbol{u} = (u_{\mathrm{loc},1}, \ldots, u_{\mathrm{loc},n_p}, u_{\mathrm{glob}}) : \Omega \to \mathbb{R}^{n_p+1}.$$

Global variations are typically assumed to be uncorrelated with respect to the local ones. The latter, however, are known to have high spatial correlations. Similarly to the treatment in Chapter 4, we model these correlations using the composite correlation function given in Equation 4.4, which is inspired by the variation patterns induced by the fabrication process [12, 15, 41]. Specifically, the correlation function imposes similarities between those locations on the die that are close to each other as well as between those locations that are at the same distance from the center of the die; see also [7, 43, 44, 51, 69].

Although Equation 4.4 captures certain features that are characteristic of the fabrication process, it is still an idealization. In practice, it can be difficult to make a justifiable choice and tune such a formula, which is a prerequisite for techniques based on the continuous KL decomposition, such as those discussed in Section 5.4. A correlation matrix, on the other hand, can be readily estimated from measurements and thus is a more probable input to probabilistic analysis. Hence, we use Equation 4.4 for the sole purpose of constructing a

correlation matrix of $\{u_{\mathrm{loc},i}\}_{i=1}^{n_p}$. For convenience, this correlation matrix is extended by one dimension in order to accommodate u_{glob} along with $\{u_{\mathrm{loc},i}\}_{i=1}^{n_p}$. Thus, the matrix acquires one additional nonzero diagonal element equal to unity; the resulting matrix is the correlation matrix of $\boldsymbol{u}$ denoted by $\mathbb{C}\mathrm{orr}(\boldsymbol{u})$.

Let us now be more specific about the power model in Equation 5.9. In the ongoing scenario, f can be rewritten as the following summation:

$$f(i, \boldsymbol{q}, \boldsymbol{u}) = f_{\mathrm{dyn}}(i) + f_{\mathrm{stat}}(\boldsymbol{q}, \boldsymbol{u})$$

where $f_{\mathrm{dyn}} : \mathbb{N}_+ \to \mathbb{R}^{n_p}$ and $f_{\mathrm{stat}} : \mathbb{R}^{n_p} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_p}$. Without loss of generality, the dynamic component f_{dyn} is assumed to be given as a dynamic power profile (recall Equation 2.1) denoted by $\mathbf{P}_{\mathrm{dyn}}$. Similarly to Section 4.6, the modeling of the static component f_{stat} is based on SPICE simulations of a reference electrical circuit composed of BSIM4 devices [106] configured according to the 45-nm PTM HP model [108]; specifically, we use a series of CMOS invertors. The simulations are performed with respect to a sufficiently wide fine-grained two-dimensional grid—the effective channel length against temperature—and the results are tabulated. An interpolation technique is then utilized whenever it is necessary to calculate f_{stat} at a point within the range of the grid.

Lastly, in order to be able to perform temperature calculations, an adequate thermal RC circuit should be constructed. Given the specification of the platform under consideration—including the floorplan of the die and the configuration of the thermal package—this circuit is obtained by means of HotSpot [100]. The structure of the circuit is the one described in Section 2.3.

To conclude, in this section, we address the variability in the effective channel length. The input to our analysis is composed of the marginal distributions of the uncertain parameters $\boldsymbol{u}$, which are beta distributions, and the corresponding correlation matrix $\mathbb{C}\mathrm{orr}(\boldsymbol{u})$. Let us now go over the other stages of our methodology presented in Section 5.6 and depicted in Figure 5.2.

5.8.2 Probability Transformation

At Stage 2 in Figure 5.2, $\boldsymbol{u}$ should be processed in order to extract a vector of mutually independent random variables $\boldsymbol{z}$ via a suitable transformation $\mathbb{T}$; see Equation 5.1. Following the guidance given in Section 5.6.2, the most apposite $\mathbb{T}$ in the ongoing scenario is the Nataf transformation. The whole procedure is described in detail in Appendix A.4 and can be seen in Equation A.8.

Using this specific $\mathbb{T}$, arbitrary marginal distributions can be prescribed for $\boldsymbol{z}$. There are no restrictions in this regard as long as a suitable polynomial basis can be constructed, which is discussed in Section 5.6.3. We let $\boldsymbol{z}$ have beta distributions, keeping $\boldsymbol{u}$ and $\boldsymbol{z}$ in the same family of distributions.

Since the number of stochastic dimensions, which is $n_u = n_p + 1$ in the case of $\boldsymbol{u}$, directly impacts the computational cost of PC expansions, which is noted in Section 5.6.3, one should consider the possibility of model order reduction

before constructing these expansions. Therefore, the reduction procedure described in Appendix A.4 in connection with $\mathbb{T}$ is assumed to be engaged in this transformation. The reduced dimensionality is denoted by n_z.

5.8.3 Surrogate Construction

At Stage 3 in Figure 5.2, the uncertain parameters, power model, and temperature model developed in the previous subsections are to be fused together under the desired workload $\mathbf{P}_{\text{dyn}}$ in order to produce the corresponding stochastic power and temperature profiles denoted by $\mathbf{P}$ and $\mathbf{Q}$, respectively.

In the current scenario, the construction of PC expansions is based on the Jacobi polynomial basis, since it is preferable in situations involving beta-distributed parameters [120]. To give a concrete example, for a dual-core platform ($n_p = 2$) with two stochastic dimensions ($n_z = 2$), the second-level PC expansion ($l_c = 2$) of temperature at time step i is as follows:

$$\begin{aligned}\mathcal{C}_2^2(\boldsymbol{q}_i) = \hat{\boldsymbol{q}}_{i,(0,0)}\psi_{(0,0)} + \hat{\boldsymbol{q}}_{i,(1,0)}\psi_{(1,0)} + \hat{\boldsymbol{q}}_{i,(0,1)}\psi_{(0,1)} \\ + \hat{\boldsymbol{q}}_{i,(1,1)}\psi_{(1,1)} + \hat{\boldsymbol{q}}_{i,(2,0)}\psi_{(2,0)} + \hat{\boldsymbol{q}}_{i,(0,2)}\psi_{(0,2)}\end{aligned} \tag{5.13}$$

where the coefficients $\{\hat{\boldsymbol{q}}_{ij}\}$ are vectors with two elements corresponding to the two processing elements. Regarding the basis,

$$\begin{aligned}\psi_{(0,0)}(\boldsymbol{z}) &= 1,\\ \psi_{(1,0)}(\boldsymbol{z}) &= 2z_1,\\ \psi_{(0,1)}(\boldsymbol{z}) &= 2z_2,\\ \psi_{(1,1)}(\boldsymbol{z}) &= 4z_1z_2\\ \psi_{(2,0)}(\boldsymbol{z}) &= \frac{15}{4}z_1^2 - \frac{3}{4}, \text{ and}\\ \psi_{(0,2)}(\boldsymbol{z}) &= \frac{15}{4}z_2^2 - \frac{3}{4}.\end{aligned}$$

The Jacobi polynomials have two parameters [120], and the ones shown above correspond to the case where both parameters are equal to two. Such a series can be shorter or longer, depending on the accuracy requirements given by l_c. The expansion of power has the same structure but different coefficients.

The next step is to compute the coefficients of power $\{\hat{\boldsymbol{p}}_{ij}\}$ in Equation 5.11, which subsequently yield the coefficients of temperature $\{\hat{\boldsymbol{q}}_{ij}\}$. As discussed in Section 5.6.3, these computations involve multidimensional integration with respect to the distribution of $\boldsymbol{z}$, and they should be performed numerically using an adequate quadrature $\mathcal{Q}_{l_q}^{n_z}$. When beta distributions are involved, the natural choice is Gauss–Jacobi quadratures, which belong to the class of Gaussian quadratures introduced in Appendix A.5. Given $\mathcal{Q}_{l_q}^{n_z}$, the coefficients are computed as shown in Equation 5.3. It is important to note that l_q should be chosen in such a way that the quadrature is exact for polynomials of total order

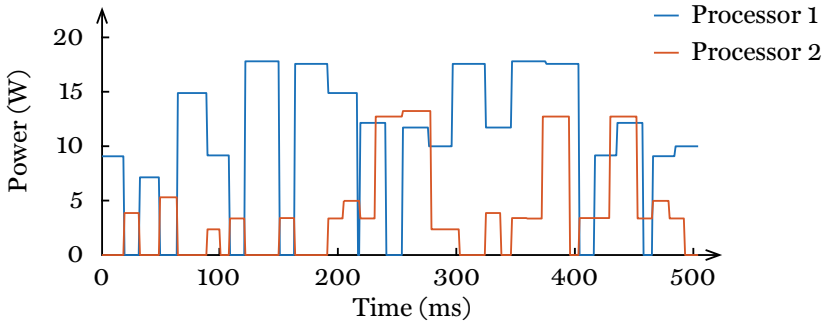

Figure 5.4: Example of a dynamic power profile of a dual-core platform

up to at least $2l_c$, that is, twice the level of PC expansions, which is discussed in Section 5.6.3. Consequently, $l_q \geq l_c$, since the quadrature is Gaussian.

To summarize, we have completed four out of five stages of the proposed framework depicted in Figure 5.2. The result is a lightweight surrogate for the entire system. At each time step, the surrogate is composed of two n_p-valued polynomials—one is for power, and the other one for temperature—which are defined in terms of n_z mutually independent random variables.

5.8.4 Post-Processing

At Stage 4 in Figure 5.2, the constructed expansions are utilized in order to assist the designer in analyzing the impact of process variation on power- and temperature-related characteristics of the system that is being developed. Consider, for example, Equation 5.13. It can be seen that the surrogate model has a negligibly small computational cost: for any outcome of z, one can readily calculate the corresponding temperature by plugging this outcome into Equation 5.13; the same applies to power. Therefore, the representation can be trivially analyzed in order to retrieve various statistics about the system. Let us illustrate a few of them using the expansion given in Equation 5.13.

Assume that the dynamic power profile $\mathbf{P}_{\text{dyn}}$ is the one shown in Figure 5.4. Having constructed a surrogate with respect to this profile, we can calculate, for instance, the expectation and variance of the temperature that the system has at a certain moment in time, which is a trivial operation given the formulae in Equation 5.8. For the whole time span of $\mathbf{P}_{\text{dyn}}$, these quantities are plotted in Figure 5.5 where the dashed lines correspond to one standard deviation above the corresponding expectations. The displayed curves closely match those obtained via MC sampling with $n_\omega = 10^4$ samples; however, our method takes less than a second, whereas MC sampling takes more than a day, which will be discussed further in Section 5.9. In addition, the PDF of the temperature at that moment can be estimated. This operation is performed by

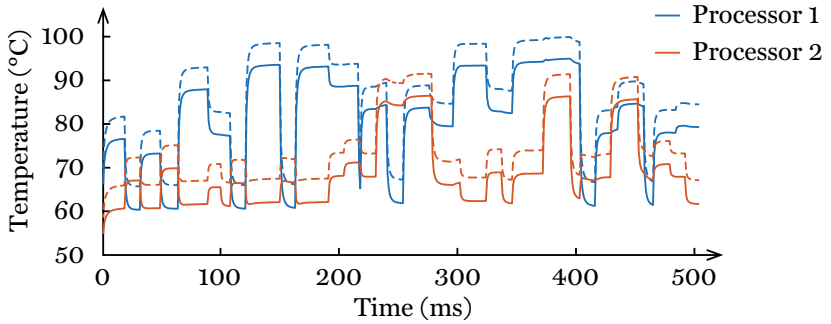

Figure 5.5: Expectation (*solid*) and one standard deviation above it (*dashed*) of a stochastic temperature profile of a dual-core platform

sampling the surrogate, in which case we might obtain curves similar to those shown in Figure 5.6, which is a part of a different example given in Section 5.9.

5.9 Transient Analysis: Experimental Results

In this section, we evaluate our framework using different configurations of the illustrative application. All the experiments are conducted on a GNU/Linux machine equipped with an Intel Core i7 2.66 GHz and 8 GB of RAM. All the configuration files used in the experiments are available online at [18].

Let us first elaborate on the default configuration of our setup, which is subsequently adjusted according to the purpose of each particular experiment. We consider a 45-nm technological process. The effective channel length is assumed to have a nominal value of 17.5 nm [108] and a standard deviation of 2.25 nm where the global and local variations are equally weighted. The correlation matrix of the uncertain parameters is computed based on Equation 4.4 where the length-scale parameters ℓ_{SE} and ℓ_{OU} are set to half the size of the square die. In the model order reduction described in Appendix A.4, the threshold parameter η is set to 0.99, which preserves 99% of the variance of the data. The floorplan of the platform is constructed in such a way that the processing elements form a regular grid. The dynamic power profiles involved in the experiments are based on simulations of applications that are randomly generated via TGFF [34]. The time step Δt of power and temperature profiles is set to 1 ms, which is also the time step of the recurrence in Equation 5.11.

The construction of PC expansions and integration grids follows the exposition given in Section 5.6.3. The tuning parameter γ in Equation 5.5 is set to zero, which turns the anisotropic index set in Equation 5.4 into the isotropic one in Equation A.12. Anisotropy will be exploited in a later section.

In the following, we focus on the assessment of temperature profiles. Note, however, that the results for temperature allow one to draw reasonable con-

clusions about the performance of the proposed framework with respect to power, since power is an intermediate step toward temperature, and any accuracy problems with respect to power are expected to propagate to temperature.

Additionally, it is worth noting that, since the temperature-driven studies discussed in Section 5.4, namely [51, 58, 59, 69], work under the static steady-state assumption (the work in [59] is also limited to the maximum temperature, and the one in [51] does not model the power-temperature interplay), a one-to-one comparison with the proposed technique is not possible.

For purposes of comparison, we employ MC sampling, which is introduced in Section 1.5. This approach samples the quantity of interest directly, meaning that there is no intermediate representation involved in these calculations. There is no model order reduction applied prior to direct sampling, which preserves the whole variance of the problem at hand, and the system in Equation 2.5 is solved using traditional techniques, namely the fourth- and fifth-order Runge–Kutta formulae (the Dormand–Prince method) [87].

5.9.1 Approximation Accuracy

The first set of experiments aims to investigate the accuracy of our framework with respect to direct sampling. At this point, it is important to realize that the true distributions of temperature are unknown, and both the PC and MC approaches introduce errors. These errors decrease as the level l_c of PC expansions and the number of samples n_ω of MC sampling increase. Hence, instead of postulating that the MC technique with a certain number of samples is the solution that we should achieve, we vary both l_c and n_ω and monitor the corresponding difference between the results produced by the two alternatives.

We also study the impact of the correlation structure between the local random variables $\{u_{\mathrm{loc},i}\}_{i=1}^{n_p}$; recall Section 5.8. More specifically, apart from l_c and n_ω, we change the balance between the two correlation functions shown in Equation 4.4, that is, the squared-exponential kernel k_{SE} and the Ornstein–Uhlenbeck kernel k_{OU}, which is controlled by the weight coefficient $w \in [0, 1]$.

The PC and MC methods are compared using three error metrics. The first two are the normalized root-mean-square errors (NRMSEs) of the expectation and variance of temperature profiles. The third metric is the mean of the NRMSEs of the empirical PDFs of temperature constructed for all processing elements at all time steps. The metrics are denoted by $\epsilon_{\mathbb{E}}$, $\epsilon_{\mathbb{V}\mathrm{ar}}$, and ϵ_f, respectively. The first two are straightforward to interpret, and they are calculated using the analytical expressions in Equation 5.8. The third one is a strong indicator of the quality of the distributions estimated by our framework, and it is computed by sampling the constructed PC expansions. In contrast to direct sampling, this sampling incurs negligible overhead, as noted in Section 5.6.4.

The studied values for l_c, n_ω, and w are the sets $\{i\}_{i=1}^{7}$, $\{10^i\}_{i=2}^{5}$, and $\{0, 0.5, 1\}$, respectively. The three variants of w correspond to the total dominance of k_{OU} ($w = 0$), perfect balance between k_{SE} and k_{OU} ($w = 0.5$), and

Table 5.1: Accuracy of the proposed solution and Monte Carlo sampling when the Ornstein–Uhlenbeck kernel dominates

l_c/n_ω	$\epsilon_{\mathbb{E}}$ (%) 10^2	$\epsilon_{\mathbb{E}}$ (%) 10^3	$\epsilon_{\mathbb{E}}$ (%) 10^4	$\epsilon_{\mathbb{E}}$ (%) 10^5	$\epsilon_{\mathbb{V}\mathrm{ar}}$ (%) 10^2	$\epsilon_{\mathbb{V}\mathrm{ar}}$ (%) 10^3	$\epsilon_{\mathbb{V}\mathrm{ar}}$ (%) 10^4	$\epsilon_{\mathbb{V}\mathrm{ar}}$ (%) 10^5	ϵ_f (%) 10^2	ϵ_f (%) 10^3	ϵ_f (%) 10^4	ϵ_f (%) 10^5
1	1.70	0.92	0.51	0.48	88.19	55.73	55.57	53.08	10.88	11.48	8.85	8.83
2	1.36	0.58	0.20	0.18	67.66	23.30	23.05	19.64	10.15	10.11	6.26	6.04
3	1.26	0.49	0.15	0.14	61.16	13.06	12.78	9.08	5.49	5.04	2.95	2.73
4	1.23	0.45	0.14	0.14	58.49	8.85	8.57	4.78	3.84	2.02	1.50	1.51
5	1.21	0.44	0.14	0.14	57.31	7.00	6.71	2.92	3.83	2.27	1.03	0.84
6	1.21	0.44	0.14	0.14	56.75	6.12	5.83	2.08	3.08	1.94	0.93	0.66
7	1.20	0.43	0.14	0.14	56.41	5.60	5.31	1.62	2.78	1.39	0.72	0.62

Table 5.2: Accuracy of the proposed solution and Monte Carlo sampling when the correlation kernels are balanced

l_c/n_ω	$\epsilon_{\mathbb{E}}$ (%) 10^2	$\epsilon_{\mathbb{E}}$ (%) 10^3	$\epsilon_{\mathbb{E}}$ (%) 10^4	$\epsilon_{\mathbb{E}}$ (%) 10^5	$\epsilon_{\mathbb{V}\mathrm{ar}}$ (%) 10^2	$\epsilon_{\mathbb{V}\mathrm{ar}}$ (%) 10^3	$\epsilon_{\mathbb{V}\mathrm{ar}}$ (%) 10^4	$\epsilon_{\mathbb{V}\mathrm{ar}}$ (%) 10^5	ϵ_f (%) 10^2	ϵ_f (%) 10^3	ϵ_f (%) 10^4	ϵ_f (%) 10^5
1	1.66	0.98	0.60	0.57	65.82	64.11	66.13	66.70	10.97	10.69	9.27	8.77
2	1.31	0.63	0.27	0.23	49.55	29.21	30.49	28.24	6.43	5.42	3.87	3.59
3	1.13	0.44	0.16	0.14	43.44	15.94	16.88	13.48	5.60	3.80	1.83	1.53
4	1.17	0.48	0.17	0.14	40.24	9.11	9.80	5.71	5.48	3.80	1.77	1.47
5	1.07	0.38	0.16	0.16	39.68	7.96	8.56	4.35	3.80	1.72	1.59	1.62
6	1.19	0.49	0.18	0.15	38.23	5.19	5.51	1.24	4.62	2.86	1.16	0.86
7	0.99	0.30	0.21	0.21	38.27	5.27	5.59	1.29	3.45	2.01	1.82	1.68

Table 5.3: Accuracy of the proposed solution and Monte Carlo sampling when the squared-exponential kernel dominates

l_c/n_ω	$\epsilon_{\mathbb{E}}$ (%) 10^2	$\epsilon_{\mathbb{E}}$ (%) 10^3	$\epsilon_{\mathbb{E}}$ (%) 10^4	$\epsilon_{\mathbb{E}}$ (%) 10^5	$\epsilon_{\mathbb{V}\mathrm{ar}}$ (%) 10^2	$\epsilon_{\mathbb{V}\mathrm{ar}}$ (%) 10^3	$\epsilon_{\mathbb{V}\mathrm{ar}}$ (%) 10^4	$\epsilon_{\mathbb{V}\mathrm{ar}}$ (%) 10^5	ϵ_f (%) 10^2	ϵ_f (%) 10^3	ϵ_f (%) 10^4	ϵ_f (%) 10^5
1	1.49	0.27	0.15	0.15	44.86	42.41	43.10	46.51	12.45	11.19	9.57	9.21
2	1.42	0.22	0.17	0.14	26.47	8.89	1.56	5.03	11.84	6.22	5.52	4.79
3	1.40	0.21	0.19	0.15	24.90	7.54	4.08	1.39	10.62	2.93	1.66	1.42
4	1.45	0.24	0.16	0.14	24.78	7.57	4.50	1.27	9.92	1.98	0.72	0.40
5	1.38	0.20	0.20	0.16	25.14	7.63	3.41	1.77	10.19	1.90	0.78	0.70
6	1.48	0.25	0.15	0.14	24.64	7.58	4.93	1.34	9.90	2.27	1.14	0.74
7	1.34	0.19	0.23	0.19	24.86	7.48	4.13	1.40	8.47	1.57	1.13	1.24

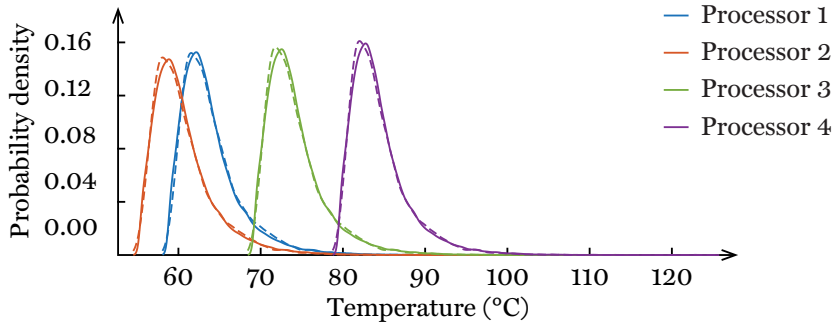

Figure 5.6: Example of probability density functions computed using the proposed solution (*solid*) and Monte Carlo sampling (*dashed*)

total dominance of k_{SE} ($w = 1$). A comparison for a quad-core architecture with a dynamic power profile of $n_s = 10^2$ steps is given in Table 5.1, Table 5.2, and Table 5.3, which correspond to $w = 0$, $w = 0.5$, and $w = 1$, respectively. Each table contains three subtables: the left one is for $\epsilon_{\mathbb{E}}$, the middle one for $\epsilon_{\mathbb{V}\mathrm{ar}}$, and the right one for ϵ_f, which results in nine subtables in total.

The columns of the tables that correspond to high values of n_ω can be used to assess the accuracy of the constructed PC expansions; likewise, the rows that correspond to high values of l_c can be used to gauge the sufficiency of the number of MC samples. One can immediately note that, in all the subtables, all the error metrics tend to decrease from the top-left corners (low values of l_c and n_ω) to the bottom-right corners (high values of l_c and n_ω), which suggests that the PC and MC methods converge. There are a few outliers associated with low expansion levels and the random nature of sampling. For instance, $\epsilon_{\mathbb{V}\mathrm{ar}}$ increases from 66.13 to 66.7 and ϵ_f from 1.59 to 1.62 when n_ω makes a transition from 10^4 to 10^5 in Table 5.2. However, the main trend is still clear.

For clarity of the discussions below, we focus primarily on one of the three tables, namely Table 5.2. The conclusions drawn with respect to Table 5.2 will be generalized to the other two tables at the end of this subsection.

First, we assess the accuracy of our technique and thus pay particular attention to the columns of Table 5.2 corresponding to high values of n_ω. It can be seen that the error of the expectation is small even when $l_c = 1$. Concretely, it is bounded by 0.6%; see $\epsilon_{\mathbb{E}}$ for $l_c \geq 1$ and $n_\omega \geq 10^4$. The error of the variance starts at 66.7% for the first-level PC expansions and drops significantly to 5.71% and below for the fourth level and higher; see $\epsilon_{\mathbb{V}\mathrm{ar}}$ for $l_c \geq 4$ and $n_\omega = 10^5$. The error of the PDF allows us to conclude that the PDFs computed by means of PC expansions starting from the third level closely follow those estimated by the MC technique with a large number of samples. The observed difference in Table 5.2 is bounded by 1.83%; see ϵ_f for $l_c \geq 3$ and $n_\omega \geq 10^4$.

In order to give a better sense of the proximity of the two methods, Figure 5.6 shows the PDFs computed using our framework with $l_c = 4$ (the solid lines) along with those calculated by the MC approach with $n_\omega = 10^4$ (the dashed lines) at time 50 ms. It can be seen that the PDFs closely match each other. Note that this example concerns one particular time step. Such curves are readily available for the other steps of the analyzed time span as well.

Next, we investigate the convergence of the MC technique and consequently watch the rows of Table 5.2 that correspond to PC expansions of high levels. Similarly to the previous observations, even for low values of n_ω, the error of the expectation estimated by direct sampling is relatively small. Specifically, this error is bounded by 1.19%; see $\epsilon_{\mathbb{E}}$ for $l_c \geq 4$ and $n_\omega = 10^2$. At the same time, the case with $n_\omega = 10^2$ has high error rates in terms of the variance and PDF: they are above 38% and around 4%, respectively; see $\epsilon_{\mathbb{V}\mathrm{ar}}$ and ϵ_f for $l_c \geq 4$ and $n_\omega = 10^2$. The results in the cases with $n_\omega \geq 10^3$ are reasonably more accurate, and the one with $n_\omega = 10^4$ appears to be sufficiently adequate.

The above conclusions drawn with respect to the results reported in Table 5.2 ($w = 0.5$) are directly applicable to those reported in Table 5.1 ($w = 0$) and Table 5.3 ($w = 1$). The only difference is that the average error rates are lower when either of the two correlation kernels shown in Equation 4.4 dominates. In particular, according to $\epsilon_{\mathbb{V}\mathrm{ar}}$, the case with $w = 1$, which corresponds to k_{SE} and is reported in Table 5.3, stands out as the least challenging.

Guided by the above observations, we conclude that our framework delivers sufficiently accurate results starting from $l_c = 4$, and that the MC estimates can be considered to be sufficiently reliable starting from $n_\omega = 10^4$.

5.9.2 Computational Speed

The second set of experiments is to measure the speed of our framework with respect to direct sampling. To this end, we keep l_c and n_ω fixed and equal to 4 and 10^4, respectively. The latter value is also similar to those used in the literature [7, 44, 51, 58, 69, 99, 118] and consistent with the theoretical results on the accuracy of MC sampling presented in [33]. Analogously to the previous subsection, we report the results obtained for different values of the weight coefficient w, which impacts the number of the independent variables $\boldsymbol{z} : \Omega \to \mathbb{R}^{n_z}$ preserved after the model order reduction described in Appendix A.4.

First, we vary the number of processing elements n_p, which directly affects the dimensionality of the uncertain parameters $\boldsymbol{u} : \Omega \to \mathbb{R}^{n_u}$; recall Section 5.8. The results, including the dimensionality n_z of $\boldsymbol{z}$, are given in Table 5.4 where $n_p \in \{2^i\}_{i=1}^{5}$ and $n_s = 10^3$. It can be seen that the variation patterns inherent in the fabrication process [15] offer significant potential for model order reduction: n_z is observed to be at most 12, whereas the maximum number without reduction is 33 (1 global variable and 32 local ones corresponding to the case with $n_p = 32$). The amount of reduction also depends on the floorplan, which is illustrated by the decrease in n_z when n_p increases

Table 5.4: Computational speed of the proposed solution and Monte Carlo sampling with respect to the number of processing elements

	n_p	n_z	PC expansion (s)	MC sampling (h)	Speedup (×)
$w = 0$	2	2	0.16	38.77	8.76×10^5
	4	2	0.16	39.03	8.70×10^5
	8	3	0.27	39.22	5.29×10^5
	16	4	0.83	40.79	1.77×10^5
	32	7	11.02	43.25	1.41×10^4
$w = 0.5$	2	3	0.21	38.72	6.54×10^5
	4	5	0.62	38.92	2.28×10^5
	8	6	1.46	40.20	9.94×10^4
	16	10	23.53	41.43	6.34×10^3
	32	12	100.10	43.05	1.55×10^3
$w = 1$	2	3	0.20	38.23	6.88×10^5
	4	5	0.56	38.48	2.49×10^5
	8	7	2.47	39.12	5.71×10^4
	16	11	40.55	41.02	3.64×10^3
	32	8	21.41	43.82	7.37×10^3

Table 5.5: Computational speed of the proposed solution and Monte Carlo sampling with respect to the number of time steps

	n_s	PC expansion (s)	MC sampling (h)	Speedup (×)
$w = 0$	10	0.01	0.51	1.77×10^5
	10^2	0.02	3.87	7.64×10^5
	10^3	0.16	38.81	8.72×10^5
	10^4	1.58	387.90	8.84×10^5
	10^5	15.85	3877.27	8.81×10^5
$w = 0.5$	10	0.02	0.39	6.10×10^4
	10^2	0.07	3.84	2.08×10^5
	10^3	0.52	38.41	2.66×10^5
	10^4	5.31	383.75	2.60×10^5
	10^5	54.27	3903.28	2.59×10^5
$w = 1$	10	0.02	0.39	6.15×10^4
	10^2	0.07	3.88	2.05×10^5
	10^3	0.54	38.86	2.60×10^5
	10^4	5.31	390.95	2.65×10^5
	10^5	53.19	3907.48	2.64×10^5

from 16 to 32 for $w = 1$. To elaborate, one floorplan is a four-by-four grid, a square, while the other an eight-by-four grid, a rectangle. Since both are fitted into square dies, the former is spread across the whole area, whereas the latter is concentrated along the middle line; the rest is due to the peculiarities of k_{SE}.

On average, the k_{OU} kernel ($w = 0$) requires the smallest number of variables, while the combination of the k_{SE} and k_{OU} kernels ($w = 0.5$) requires the largest. This means that, in the latter case, more variables should be preserved in order to retain 99% of the variance. Consequently, the case with $w = 0.5$ is the most demanding in terms of computation time. Note that, since the results reported in the previous subsection correspond to the case with $n_p = 4$, n_z is two, five, and five in Table 5.1, Table 5.2, and Table 5.3, respectively.

At this point, it is important to realize the following. First, since the curse of dimensionality arguably constitutes the major concern of PC expansions, the applicability of our framework depends primarily on how this curse manifests itself with regard to the problem at hand, that is, on the dimensionality n_z of $\boldsymbol{z}$. Second, since $\boldsymbol{z}$ is a result of a procedure that depends on many factors, the relationship between $\boldsymbol{u}$ and $\boldsymbol{z}$ is not straightforward, which is also illustrated in the previous paragraphs. Thus, n_u can be misleading when reasoning about the applicability of our technique; n_z is well suited for this purpose.

Another observation with respect to Table 5.4 is the shallow slope of the execution time of the MC technique, which illustrates the well-known fact that the workload per sample is independent of the number of stochastic dimensions. On the other hand, the rows with $n_z \geq 10$ hint at the curse of dimensionality, which PC expansions suffer from. However, even in high dimensions, the proposed framework significantly outperforms direct sampling. For instance, in order to analyze a dynamic power profile with 10^3 steps of a platform with 32 processing elements, the MC approach requires more than 40 hours, whereas our framework takes less than two minutes; see the case with $w = 0.5$.

Next, we investigate the scaling properties of the proposed framework with respect to the duration of the analyzed time span, which is directly proportional to the number of time steps n_s covered by power and temperature profiles. The results for a quad-core platform are given in Table 5.5. Due to the long execution times demonstrated by the MC approach, its statistics for high values of n_s are extrapolated based on a smaller number of samples $n_\omega < 10^4$. Similarly to Table 5.4, we observe a certain dependency of the constructed expansions on the dimensionality n_z, which is two for $w = 0$ and five for $w = 0.5$ and $w = 1$; see Table 5.4 for $n_p = 4$. It can be seen in Table 5.5 that the computation times of both techniques grow linearly with respect to n_s, which is expected. However, our framework displays a superior performance, being up to five orders of magnitude faster than the MC alternative.

It is worth noting that the observed speedups are due to two major factors. First, PC expansions are generally superior to MC sampling when the curse of dimensionality is suppressed [36, 120], which we accomplish by model order reduction and efficient integration schemes. The second reason is the partic-

Algorithm 5.2: Calculation of dynamic steady-state power and temperature profiles given an outcome of the uncertain parameters

Input: $\boldsymbol{u} \in \mathbb{R}^{n_u}$
Output: $\mathbf{P} \in \mathbb{R}^{n_p \times n_s}$, $\mathbf{Q} \in \mathbb{R}^{n_p \times n_s}$
1: $\mathbf{Q} \leftarrow \mathbf{Q}_{\text{amb}}$ // a matrix version of $\boldsymbol{q}_{\text{amb}}$
2: **repeat**
3: $\quad \mathbf{P} \leftarrow f(\boldsymbol{u}, \mathbf{Q})$ // for all time steps at once
4: $\quad \mathbf{Q} \leftarrow$ Algorithm 3.1($\mathbf{P}$)
5: **until** a stopping condition is satisfied
6: **return** $\mathbf{P}$, $\mathbf{Q}$

ular technique used in the framework for solving the temperature model and constructing PC expansions in a stepwise manner shown in Equation 5.11.

5.10 Dynamic Steady-State Analysis

In this section, we discuss dynamic steady-state power and temperature analysis under process variation. Unlike transient analysis, this analysis cannot be done one time step at a time, since the repetitive workload needs to be taken into account at once in order to calculate the corresponding dynamic steady state. Thus, similarly to the contrast between Section 3.2 and Section 3.4, the solutions in Section 5.7 and this section differ. However, they still rely on the same methodology outlined in Section 5.6 and shown in Figure 5.2.

5.10.1 Problem Formulation

The system model is the same as the one in Section 5.7. The only difference is that it is more convenient to define the power model as follows:

$$\mathbf{P} = f(\boldsymbol{u}, \mathbf{Q}). \tag{5.14}$$

The function $f : \mathbb{R}^{n_u} \times \mathbb{R}^{n_p \times n_s} \rightarrow \mathbb{R}^{n_p \times n_s}$ is supposed to return the periodic power profile that corresponds to the periodic workload being analyzed.

The solution to probabilistic dynamic steady-state analysis is based on the deterministic one presented in Section 3.4.2. The power, temperature, and other vectors that appear in Section 3.4.2 become stochastic in the present context, which also concerns the boundary condition given in Equation 3.7. The pseudocode for a procedure that delivers $\mathbf{Q}$ for a fixed $\boldsymbol{u}$ is listed in Algorithm 3.1. The algorithm does not take account of the interdependence between power and temperature; however, this interdependence can be addressed via one of the techniques presented in Section 3.5. In this section, we use the iterative approach illustrated in Algorithm 3.2. For clarity, this algorithm is rewritten here as shown in Algorithm 5.2; the main difference is that the calculation of power on line 3 is now based on Equation 5.14.

Remark 5.3. *The application of the linear approximation described in Section 3.5 is problematic in this case. This technique is suitable when the only varying parameter is temperature, and all other parameters have nominal values. In that case, it is relatively easy to decide on a representative temperature range and apply a curve-fitting procedure. In this case, however, the power model has multiple parameters that range far from their nominal values.*

To recapitulate, the quantity of interest g in Figure 5.2 is the dynamic steady-state power and temperature profiles, which are denoted by $\mathbf{P}$ and $\mathbf{Q}$, respectively. The calculation of this quantity is shown in Algorithm 5.2.

5.10.2 Surrogate Construction

At Stage 3 in Figure 5.2, the procedure delineated in Section 5.6.3 is applied to Algorithm 5.2 from Stage 1 with respect to the output of Stage 2. Concretely, Algorithm 5.2 is utilized inside Algorithm 5.1 via Algorithm G. In this case, Algorithm G calls Algorithm 5.2 and returns $\mathbf{P}$ or $\mathbf{Q}$ or both, depending on what is actually needed for the subsequent calculations, in a suitable format.

Suppose, for instance, that the designer is interested in analyzing solely the dynamic steady-state temperature profile $\mathbf{Q}$. Following Remark 5.1 concerning vector-valued quantities, g is treated as an $n_p n_s$-element row vector, in which case each coefficient $\hat{g}_i$ in Equation 5.2 is also such a vector. The projection matrix, which is defined in Equation 5.6, and Algorithm 5.1 should be reinterpreted accordingly: $\boldsymbol{g}$ is an $n_q \times n_p n_s$ matrix whose row j is $\mathbf{Q}$ computed at point j of the quadrature in Equation 5.3 and reshaped into a row vector. Similarly, $\hat{\boldsymbol{g}}$ should be understood as an $n_c \times n_p n_s$ matrix whose row i is coefficient i of the expansion in Equation 5.2. Recall that a certain ordering is assumed to be imposed on quadrature points and polynomial terms.

The constructed PC expansion can now be post-processed as required, which is already the topic of Stage 4 in Figure 5.2; see Section 5.6.4.

5.11 Reliability Analysis

Our goal in this section is to build a flexible and computationally efficient technique for reliability analysis of electronic systems that are affected by process variation. The development is based on the general reliability model $R(\cdot|\boldsymbol{g})$ described in Section 2.4, which is not aware of process variation yet. Note also that, in this section, we address not only process uncertainty but also aging uncertainty, which is introduced in Section 1.1.3, since the latter can be more adequately mitigated when the former is accounted for in reliability analysis.

5.11.1 Problem Formulation

Our work in this context is motivated by the following two observations.

First, as underscored throughout the thesis, temperature is the driving force of many failure mechanisms. The most prominent examples include electromigration, time-dependent dielectric breakdown, stress migration, and thermal cycling; the interested reader is referred to [37] for an overview. All of these mechanisms have strong dependencies on temperature. At the same time, temperature is closely related to process parameters—such as the effective channel length and gate oxide thickness—and can vary dramatically when these parameters deviate from their nominal values. Despite these concerns, the current state-of-the-art techniques for reliability analysis of electronic systems lack a systematic treatment of process variation and, in particular, the effect of this variation on temperature, which is also the case in Section 2.4.

Second, having established a reliability model $R(\cdot|\boldsymbol{g})$ of the system under consideration, the major portion of the associated computation time is ascribed to the evaluation of the parameterization $\boldsymbol{g}$ rather than to the model *per se*, that is, for a given $\boldsymbol{g}$. For instance, $\boldsymbol{g}$ often contains estimates of the mean time to failure (MTTF) of each processing element for a range of stress levels. Thus, $\boldsymbol{g}$ typically involves computationally intensive simulations, including power analysis paired with temperature analysis; see Section 2.4.

Guided by the aforementioned observations, we employ the PC decomposition in order to construct a lightweight surrogate for $\boldsymbol{g}$. It is worth emphasizing that $R(\cdot|\boldsymbol{g})$ stays intact, which means that our approach does not impose any restrictions on $R(\cdot|\boldsymbol{g})$. Hence, the designer can take advantage of an arbitrary reliability model in a straightforward manner. Naturally, this also implies that modeling errors associated with the chosen $R(\cdot|\boldsymbol{g})$ can affect the quality of the results produced by our technique. Therefore, choosing an adequate reliability model for the problem at hand is the designer's responsibility.

Remark 5.4. *It is important to realize that there are two levels of probabilistic modeling here. First,* $R(\cdot|\boldsymbol{g})$ per se *is a probabilistic model describing the lifetime* L *of the system. Second, the parameterization* $\boldsymbol{g}$ *is another probabilistic model characterizing the impact of uncertainty due to process variation on the reliability model. Therefore, the overall model can be thought of as a probability distribution over probability distributions. Given an outcome of the fabrication process and thus* $\boldsymbol{g}$*, the system's lifetime remains random.*

To conclude, the quantity of interest g, which is an output of Stage 1 in Figure 5.2, is the parameters $\boldsymbol{g}$ of the reliability model under consideration.

5.11.2 Surrogate Construction

Similarly to Section 5.10, Stage 3 and Stage 4 of the framework require no particular attention in this section except for noting that Remark 5.1 should be taken into consideration if the parameterization $\boldsymbol{g}$ has multiple entries.

In conclusion, the proposed approach to reliability analysis is founded on the basis of state-of-the-art reliability models, and it enriches their modeling capabilities by seamlessly incorporating the deleterious impact of process variation. In particular, the technique allows for a straightforward propagation of uncertainty from process parameters through temperature to the lifetime of the system, which is an important application, since temperature is the driving force of many failure mechanisms. In contrast to the straightforward use of MC sampling, the lightweight surrogates that we construct make the subsequent analysis highly efficient from a computational perspective.

5.12 Energy Optimization

The framework proposed in this chapter is illustrated in the context of design-space exploration, which is analogous to the optimization in Section 3.6. Concretely, we pursue energy minimization in the presence of process variation.

5.12.1 Problem Formulation

The general setup is the same as the one described in Section 3.6.2. We continue working with thermal-cycling fatigue [37], which is discussed in Section 2.4.2 and Section 3.6. Recall that the system is exposed to a periodic or approximately periodic workload, and that the corresponding temperature profile is a dynamic steady-state temperature profile. In the deterministic case, this temperature profile can be computed as described in Section 3.4.2.

The goal of the optimization in this section is to find a schedule that minimizes the system's energy consumption while satisfying certain constraints. Specifically, our objective is as follows:

$$\min_{\mathcal{S}} \mathbb{E}(E(\mathcal{S})) \tag{5.15}$$

such that

$$\begin{aligned} &\tau(\mathcal{S}) \leq \tau_{\max}, \\ &\mathbb{P}\left(Q(\mathcal{S}) \geq q_{\max}\right) \leq \rho_{\text{burn}}, \text{ and} \\ &\mathbb{P}\left(\mathbb{E}\left(L(\mathcal{S})\right) \leq L_{\min}\right) \leq \rho_{\text{wear}} \end{aligned} \tag{5.16}$$

where

$$\begin{aligned} E(\mathcal{S}) &= \Delta t \left\|\mathbf{P}(\mathcal{S})\right\|_1 \text{ and} \\ Q(\mathcal{S}) &= \left\|\mathbf{Q}(\mathcal{S})\right\|_\infty . \end{aligned}$$

To begin with, $\mathcal{S}$ denotes a schedule (recall that a schedule includes not only the starting times of the tasks but also their mapping onto the processing elements); $\mathbf{P}$ and $\mathbf{Q}$ are the corresponding dynamic steady-state power and temperature profiles, respectively; Δt is their sampling interval; and $\|\cdot\|_1$ and

$\|\cdot\|_\infty$ are the Manhattan and uniform norms, respectively. The objective in Equation 5.15 is to minimize the expectation of the total energy consumption $E : \Omega \to \mathbb{R}$, which is a random variable, since energy depends on power, and power depends on the uncertain parameters $\boldsymbol{u}$. The constraints in Equation 5.16 concern time, temperature, and reliability. The first one constrains the end-to-end delay τ of the application by a deadline $\tau_{\max}$. The second one constrains the maximum temperature $Q : \Omega \to \mathbb{R}$ of the platform by $q_{\max}$ where ρ_{burn} is an acceptable probability of burning the chip. The third one constrains (from below) the lifetime $L : \Omega \to \mathbb{R}$ of the system by $L_{\min}$ where ρ_{wear} is an acceptable probability of premature failure due to wear.

The first constraint in Equation 5.16 is deterministic. On the other hand, the other two constraints are probabilistic, since they are based on stochastic quantities. In the very last constraint, we set an upper bound on the expectation of L. It should be noted that this expectation is a random variable *per se* due to the nested structure of the reliability model discussed in Remark 5.4.

In this section, the quantity of interest g in Figure 5.2 is the vector

$$\boldsymbol{g} = (E, Q, L) \tag{5.17}$$

where the first element corresponds to the total energy consumption; the second one is the maximum temperature; and, with a slight abuse of notation, the third one is the parameters of the reliability model characterizing the lifetime (they are denoted by $\boldsymbol{g}$ in Section 2.4 and Section 5.11). Each element implicitly depends on $\mathcal{S}$ and $\boldsymbol{u}$. In Algorithm 5.1, Algorithm G is a subroutine that transforms $\boldsymbol{z}$ into $\boldsymbol{u}$, makes a call to Algorithm 5.2, and processes the resulting power and temperature profiles as required in order to compute the above $\boldsymbol{g}$.

The construction of a surrogate at Stage 3 is standard; see Section 5.6.3. On the other hand, the post-processing stage, Stage 4, is worth discussing.

5.12.2 Post-Processing

In this context, the post-processing stage concerns the usage of expansions of Equation 5.17 inside an optimization procedure whose objective and constraints are the ones shown in Equation 5.15 and Equation 5.16, respectively.

We begin by delineating the optimization procedure itself. Similarly to Section 3.6, we make use of a genetic algorithm [97]. Each chromosome contains $2n_t$ genes that encode a mapping of the tasks onto the processing elements and a set of priorities for the tasks. The population contains $4n_t$ individuals that are initialized using uniform distributions. The parents for the next generation are chosen by a tournament selection with the number of competitors equal to 20% of n_t. A one-point crossover is then applied to 80% of the parents. Each parent undergoes a uniform mutation where each gene is altered with probability 0.01. The top 5% of individuals always survive. The stopping condition is the absence of improvement within 10 successive generations.

Let us now turn to the evaluation of the fitness of a chromosome, which is where PC expansions come into play. First, the information encoded in the chromosome is fed to a list scheduler [1], and the scheduler produces a schedule $\mathcal{S}$. We then check the timing constraint (the first one in Equation 5.16), which does not require any uncertainty analysis. If it is violated, we set the fitness to the amount of this violation relative to the constraint—that is, to the difference between the actual end-to-end delay and the deadline $\tau_{\max}$ divided by $\tau_{\max}$—and add a large constant C. If the timing constraint is satisfied, we perform our uncertainty analysis and proceed to checking the temperature and lifetime constraints. If any of them is violated, we set the fitness to the total relative amount of violation plus $C/2$. If all the constraints are satisfied, the fitness of the chromosome is set to the expected energy consumption.

The aforementioned examination of the temperature and lifetime constraints as well as the evaluation of the expected energy consumption are undertaken by means of a PC expansion of Equation 5.17 following the description given in Section 5.6.4. Specifically, the two probabilities needed in Equation 5.16 are estimated via sampling the lightweight polynomial surrogate. By contrast, the expectation in Equation 5.15 requires no sampling and is straightforwardly computed using the analytical formula shown in Equation 5.8.

Remark 5.5. *In order to speed up the optimization process, we use caching and parallel computing. The fitness value of each evaluated chromosome is stored in memory and reused whenever a chromosome with the same set of genes is encountered, and unseen (not cached) individuals are assessed in parallel.*

5.13 Energy Optimization: Illustrative Application

In order to give a better sense of the approach to reliability analysis and optimization presented in Section 5.11 and Section 5.12, we consider a concrete application, meaning that we specify the uncertain parameters and discuss the accompanying computations. This application is also utilized for the quantitative evaluation of our technique presented in the next section, Section 5.14.

5.13.1 Problem Formulation

Assume that the structure of the reliability model $R(\cdot|\boldsymbol{g})$ of the system at hand is the one given in Equation 2.6 where each individual reliability function $R_i(\cdot|\boldsymbol{g}_i)$ is the one shown in Equation 2.8 with its own parameters η_i and β_i.

During each iteration, the temperature of processing element i exhibits $n_{c,i}$ cycles. Each cycle generally has different characteristics and hence causes a different amount of damage to the processing element. This aspect is accounted for by adjusting η_i as shown in Equation 2.11. The shape parameter β_i is known to be indifferent to temperature [13]. For simplicity, assume that β_i does not depend on process parameters either, and that $\beta_i = \beta$ for $i = 1, \ldots, n_p$.

Under the above assumptions, Remark 2.1 applies, and the lifetime $L : \Omega \to \mathbb{R}$ of the system has a Weibull distribution as follows:

$$L|(\eta, \beta) \sim \text{Weibull}(\eta, \beta)$$

where η is the one given in Remark 2.1 combined with Equation 2.11. Even though the reliability model has two parameters, only one of them is uncertain to the designer, namely η. Therefore, we treat the model as if it was parameterized only by η. The shape parameter β is assumed to be implicitly given.

In the case of reliability analysis under process variation without any accompanying exploration of the design space, one can proceed to constructing a PC expansion of η. Having obtained this lightweight surrogate, the reliability of the system can be studied from various perspectives. In the current scenario, however, the quantity of interest g is the one given in Equation 5.17, since it allows for evaluating the objective function and constraints defined in Equation 5.15 and Equation 5.16, respectively. In Equation 5.17, the component denoted by L stands for the parameterization of the reliability model; consequently, it is η in the illustrative application developed in this section.

Let us now turn our attention to the uncertain parameters $\boldsymbol{u}$ of the problem being addressed. We focus on two crucial process parameters: the effective channel length and gate oxide thickness. Each processing element is then assigned two random variables corresponding to the two process parameters, which means that $n_u = 2n_p$ in the current example; see also Section 5.3.

Remark 5.6. *The variability in a process parameter at a spatial location can be modeled as a composition of several parts—such as inter-lot, inter-wafer, inter-die, and intra-die variations—which is demonstrated in Section 5.8. In this section, we illustrate a different approach. From a mathematical perspective, it is sufficient to consider only one random variable per location with an adequate distribution and correlations with respect to the other locations.*

Based on Section 5.6.1, the parameters $\boldsymbol{u}$ are assumed to be given as a set of marginal distributions and a correlation matrix denoted by $\{F_i\}_{i=1}^{n_u}$ and $\mathbb{C}\text{orr}(\boldsymbol{u})$, respectively. Note that the number of distinct marginals is only two, since n_p components of $\boldsymbol{u}$ correspond to the same process parameter.

Both process parameters, the effective channel length and gate oxide thickness, correspond to Euclidean distances; they take values on bounded intervals of the positive half of the real line. Consequently, similarly to Section 5.8, we model the two process parameters using the four-parameter family of beta distributions shown in Equation 5.12. Without loss of generality, the parameters are assumed to be independent of each other, and the correlations between those elements of $\boldsymbol{u}$ that correspond to the same process parameter are assumed to be given by the correlation function shown in Equation 4.4.

The process parameters manifest themselves in the calculations associated with the power model shown in Equation 5.14 through static power. Analogously to Section 5.8, the modeling here is based on SPICE simulations of a

series of CMOS invertors. The invertors are taken from the 45-nm open cell library by NanGate [107] and configured according to the 45-nm PTM HP model [108]. The simulations are performed on a fine-grained and sufficiently broad three-dimensional grid comprising the effective channel length, gate oxide thickness, and temperature; the results are tabulated. An interpolation algorithm is subsequently employed whenever static power is to be evaluated at a particular point within the range of the grid. The output of this model is scaled up to account for about 40% of the total power consumption [75]. Regarding temperature, the thermal RC circuit utilized for dynamic steady-state analysis is constructed by virtue of HotSpot [100] as described in Section 2.3.

At this point, the two outputs of Stage 1 are now specified.

5.13.2 Probability Transformation

At Stage 2 in Figure 5.2, the uncertain parameters $\boldsymbol{u}$ are transformed into a vector of independent random variables $\boldsymbol{z}$ via a suitable transformation $\mathbb{T}$. Specifically, we use the one given in Equation A.8, which also includes model order reduction. Unlike Section 5.8, in this section, we let $\boldsymbol{z}$ obey the standard Gaussian distribution and, therefore, tailor $\mathbb{T}$ accordingly; see Appendix A.4.

5.13.3 Surrogate Construction

Since the auxiliary variables $\boldsymbol{z} = (z_i)_{i=1}^{n_z}$ are Gaussian, the polynomial basis considered at Stage 3 is to be composed of Hermite polynomials, which is the exact scenario described in Appendix A.7. The variables also tell us how to approach numerical integration needed for evaluation of the coefficients of PC expansions: since we are interested in integrals with respect to the standard Gaussian measure, Gauss–Hermite quadratures [68] are worth considering. These quadratures are especially efficient, since they belong to the class of Gaussian quadratures and thus inherit their properties; see Appendix A.5.

Lastly, let us illustrate the Hermite basis. In the case of working with only one standard Gaussian variable ($n_z = 1$), a second-level PC expansion ($l_c = 2$) of a three-dimensional quantity of interest $\boldsymbol{g}$ is as follows:

$$\mathcal{C}_2^1(\boldsymbol{g}) = \hat{\boldsymbol{g}}_{(0)}\psi_{(0)} + \hat{\boldsymbol{g}}_{(1)}\psi_{(1)} + \hat{\boldsymbol{g}}_{(2)}\psi_{(2)}$$

where $\{\hat{\boldsymbol{g}}_{\boldsymbol{i}}\} \subset \mathbb{R}^3$,

$$\begin{aligned}\psi_{(0)}(\boldsymbol{z}) &= 1,\\ \psi_{(1)}(\boldsymbol{z}) &= z_1, \text{ and}\\ \psi_{(2)}(\boldsymbol{z}) &= z_1^2 - 1.\end{aligned}$$

At Stage 4, the expansion is post-processed as described in Section 5.12.

5.14 Energy Optimization: Experimental Results

In this section, we evaluate the performance of our approach to reliability analysis and optimization presented in Section 5.11 and Section 5.12 considering the illustrative application described in Section 5.13. The technique for dynamic steady-state analysis under process variation delineated in Section 5.10 is also a part of the assessment, since it is included in the reliability model. All the experiments are conducted on a GNU/Linux machine equipped with 16 Intel Xeon E5520 2.27 GHz processors and 24 GB of RAM. All the configuration files used in the experiments are available online at [19].

We consider a 45-nm technological process and rely on the 45-nm open cell library by NanGate [107] as explained in Section 5.13. The effective channel length and gate oxide thickness are assumed to have nominal values equal to 22.5 nm and 1 nm, respectively. Based on the International Technology Roadmap for Semiconductors [54], each parameter is assumed to deviate by up to 12% of its nominal value; the percentage is treated as three standard deviations, and the assumption should be understood accordingly. Regarding the correlation function in Equation 4.4, the weight coefficient w is set to 0.5, and the length-scale parameters ℓ_{SE} and ℓ_{OU} are set to half the size of the die. The model order reduction described in Appendix A.4 is set to preserve 95% of the variance of the problem. The parameter γ used in Equation 5.5, which controls the anisotropy of PC expansions and integration grids, is set to 0.25.

Heterogeneous platforms and periodic applications are randomly generated via TGFF [34] in such a way that the execution time of each task is uniformly distributed between 10 and 30 ms, and its dynamic power between 6 and 20 W. The floorplans of the platforms being considered are regular grids with each processing element occupying 4 mm^2. The sampling interval Δt of power and temperature profiles is set to 1 ms. The stopping condition used in Algorithm 5.2 is that the NRMSE between two successive temperature profiles becomes smaller than 1%, which typically requires 3–5 iterations.

5.14.1 Approximation Accuracy

Our first task is to evaluate the accuracy of the proposed framework. To this end, in this subsection, the quantity of interest given in Equation 5.17—which encompasses the total energy consumption, maximum temperature, and parameterization of the reliability model of the system—is considered in isolation. This quantity plays the key role in the subsequent optimization, since the optimization objective and constraints depend on it, as discussed previously.

We compare the performance of our technique with that of direct MC sampling applied to the quantity shown in Equation 5.17. The operations performed by direct sampling for one sample are the same as those performed by our framework for one quadrature point. The only difference is that no model order reduction of any kind is undertaken prior to MC sampling, which

Table 5.6: Accuracy of the proposed solution with respect to the level of polynomial chaos expansions

l_c	n_c	n_q	ϵ_E (KLD)	ϵ_Q (KLD)	ϵ_L (KLD)
1	3	5	0.0415	0.1935	0.3390
2	10	21	0.0085	0.0187	0.0320
3	22	69	0.0022	0.0025	0.0046
4	49	193	0.0017	0.0024	0.0033
5	111	589	0.0016	0.0027	0.0037

ensures that the resulting accuracy is not compromised. The number of MC samples is set to 10^4, which is a practical assumption based not only on our experience but also on the literature, as discussed in Section 5.9.

The results are displayed in Table 5.6 where we consider a quad-core platform ($n_p = 4$) with 10 randomly generated applications and vary the level of polynomial expansions l_c from one to five. The errors for the three components of $\boldsymbol{g} = (E, Q, L)$ are denoted by ϵ_E, ϵ_Q, and ϵ_L, respectively. Each error metric shows the distance between the empirical probability distributions produced by our approach and the ones produced by direct sampling. The measure of this distance is the Kullback–Leibler divergence (KLD) [42, 46] where the results of direct sampling are treated as the true ones. The KLD takes nonnegative values and reaches zero only when two distributions are equal almost everywhere [35]. In general, the errors decrease as l_c increases. This trend, however, is not monotonic for PC expansions of high levels; see ϵ_Q and ϵ_L for $l_c = 5$ in Table 5.6. This observation can be ascribed to the random nature of sampling and to the reduction procedures that we undertake in order to gain speed; they reasonably impose limitations on the accuracy that can be attained by polynomial surrogates. Table 5.6 additionally contains the number of polynomial terms n_c and the number of quadrature points n_q that correspond to each value of l_c. We have performed the above experiment for platforms with both fewer and more processing elements and have observed similar results.

Based on the figures reported in Table 5.6, we consider the results delivered by third-level PC expansions, where the KLD drops to the third decimal place for all the three quantities, sufficiently accurate. Therefore, we fix l_c—and hence l_q, as they are kept synchronized—to three for the rest of the experiments.

5.14.2 Computational Speed

Our task now is to assess the speed of the proposed solution. To this end, we consider the same setup as the one outlined in the previous subsection. Table 5.7 displays the time needed to perform one characterization of $\boldsymbol{g}$ for the number of processing elements n_p increasing from 2 to 32. Note that, in this experiment, no parallel computing is utilized. It can be seen that the computa-

Table 5.7: Computational speed of the proposed solution with respect to the number of processing elements

n_p	n_z	n_c	n_q	Time (s)	Speedup (×)
2	4	19	57	0.18	175.44
4	6	22	69	0.26	144.93
8	8	27	81	0.73	123.46
16	10	30	93	1.28	107.53
32	10	33	101	2.23	99.01

tion time ranges from a fraction of a second to around two seconds. More importantly, Table 5.7 provides information about a number of complementary quantities that are of great interest to the designer, which we discuss below.

The primary quantity to pay heed to is the number of random variables n_z preserved after the reduction procedure noted in Section 5.6.2 and described in Appendix A.4. Without this reduction, n_z would be $2n_p$, since there are two process parameters per processing element; recall Section 5.13. It can be seen in Table 5.7 that there is no reduction in the case of the dual-core platform, whereas around 80% of the stochastic dimensions are eliminated in the case of the platform with 32 processing elements. One can also note that n_z is the same for the last two platforms. The magnitude of reduction is determined solely by the assumed correlation structure (see Section 5.13) and the floorplan of the platform at hand, which we also observe and discuss in Section 5.9.

Another important quantity displayed in Table 5.7 is the number of quadrature points n_q. This number is the main indicator of the computational expense of our probabilistic analysis: it is equal to the number of times Algorithm G in Algorithm 5.1 must be executed in order to construct a PC expansion of the quantity of interest g, which, in this case, is the one in Equation 5.17. Note that n_q is very low. In order to substantiate this, the last column of Table 5.7 shows the speedup of our approach with respect to 10^4 MC samples. The proposed solution is 100–200 times faster while delivering highly accurate results, as discussed earlier. It should be noted that the comparison has been drawn based on the number of evaluation points rather than on the wall-clock time, since the relative cost of other computations is negligible.

To summarize, the proposed framework for probabilistic analysis of electronic systems under process variation has been assessed using the composite quantity given in Equation 5.17. The results shown in Table 5.6 and Table 5.7 indicate that our approach is both accurate and computationally efficient.

5.14.3 Optimization Effectiveness

In this subsection, the results of the optimization procedure formulated in Section 5.12 are reported. To reiterate, the objective is to minimize the expected

Table 5.8: Computational speed of the probabilistic and deterministic optimization procedures as well as the failure rate of the latter

	Probabilistic case	Deterministic case	
n_p	Time (min)	Time (min)	Failure (%)
2	1.07	0.67	40
4	5.38	1.99	60
8	16.65	3.89	70
16	56.23	7.54	100
32	341.08	9.26	100

energy consumption as shown in Equation 5.15 while satisfying a set of constraints on the maximum end-to-end delay, maximum temperature, and minimum lifetime as shown in Equation 5.16. For optimization, we employ a genetic algorithm and evaluate candidate solutions in parallel using 16 cores.

The goal of this experiment is to justify the following assertion: reliability analysis has to account for the effect of process variation on temperature. To this end, for each problem (a pair of a platform and an application), we run the optimization procedure twice: the first run assumes the setup discussed in this section so far, and the second run treats the objective in Equation 5.15 and the constraints in Equation 5.16 as deterministic. Specifically, the second run assumes that temperature is deterministic and can be computed using the nominal values of the process parameters. Therefore, in this deterministic case, only one execution of the system is needed in order to evaluate a chromosome's fitness. Equation 5.15 and Equation 5.16 become, respectively,

$$\min_{\mathcal{S}} E(\mathcal{S})$$

and

$$\begin{aligned} &\tau(\mathcal{S}) \leq \tau_{\max}, \\ &Q(\mathcal{S}) \geq q_{\max}, \text{ and} \\ &L(\mathcal{S}) \leq L_{\min}. \end{aligned}$$

We consider five platforms with the number of processing elements n_p taking values in $\{2^i\}_{i=1}^{5}$ and 10 applications with the number of tasks n_t equal to $20n_p$; therefore, there are 50 problems in total. In Equation 5.16, ρ_{burn} and ρ_{wear} are set to 0.01. Due to the diversity of the problems, $\tau_{\max}$, $q_{\max}$, and $L_{\min}$ are found individually for each problem in order to ensure that their values are sensible to the subsequent optimization. For instance, $q_{\max}$ ranges from 90°C to 120°C, depending on the problem. Note, however, that these three parameters stay the same in both the probabilistic and deterministic cases.

The results are reported in Table 5.8. The most important message is in the last column. *Failure* refers to the percentage of the solutions produced by

the deterministic optimization that, after being re-evaluated using our probabilistic approach (that is, after taking process variation into account), have been found to violate the probabilistic constraints given in Equation 5.16. For example, for the quad-core platform, 6 out of 10 schedules that are proposed by the deterministic approach violate the constraint on the maximum temperature or the minimum lifetime (or both) when process variation is taken into consideration. As the problem becomes more and more complex, the failure rate attains higher and higher values. With 16 and 32 processing elements (320 and 640 tasks, respectively), all the deterministic solutions violate the imposed constraints. Moreover, the difference between the acceptable 1% of burn and wear ($\rho_{\text{burn}} = \rho_{\text{wear}} = 0.01$) and the actual probability of burn and wear is found to be as high as 80% in some cases, which is unacceptable.

In addition, we take a close look at those few deterministic solutions that have passed the probabilistic re-evaluation and observe that the reported reduction in the maximum temperature and energy consumption as well as the reported increase in the lifetime are overoptimistic. To elaborate, the predictions produced by the deterministic optimization, which neglects process variation, are compared with the expected values obtained when process variation is taken into account. The comparison shows that the expected energy consumption and temperature are up to 5% higher while the expected lifetime is up to 20% shorter than the ones estimated by the deterministic approach. This aspect of the deterministic optimization can mislead the designer. Hence, when studying those characteristics of electronic systems that are concerned with power, temperature, and reliability, ignorance of the effect of process variation can severely compromise the associated design decisions, making them less profitable in the best case and dangerous in the worst case.

Let us now comment on the optimization time shown in Table 5.8. It can be observed that our framework takes from around one minute to six hours (utilizing 16 cores) in order to perform the optimization, and that the deterministic technique is 2–40 times faster. However, the price to pay when relying on the latter is quite high, as discussed above. The deterministic approach is blind-guessing with highly unfavorable odds of succeeding. Therefore, the computation time of our framework is considered reasonable and affordable.

Lastly, we perform an experiment targeted at investigating the impact of the lifetime constraint in Equation 5.16 on the reduction in the expected energy consumption. To this end, we run our probabilistic optimization (all 50 problems) without the reliability constraint and compare the corresponding results with those obtained when the lifetime constraint is included. We observe that the expected energy consumption is higher when the constraint is taken into account; however, the difference vanishes when the complexity of the problem increases. On average, the cost of the lifetime constraint is below 5% in terms of energy consumption. Without the constraint, however, no (probabilistic) guarantees on the system's lifetime can be given.

5.15 Conclusion

As emphasized throughout the thesis, electronic-system designs that ignore uncertainty are both inefficient and unreliable. Acknowledging this exigent concern, we have developed a flexible framework for analysis of electronic systems that are subject to process variation. The framework is capable of modeling diverse system-level quantities that are dependent on diverse process parameters, which, in turn, obey diverse probability distributions. Our methodology delivers a lightweight surrogate for the quantity being analyzed, thereby providing the designer with a computationally efficient means of calculating the probability distribution and other characteristics of this quantity.

Leveraging our general methodology, we have presented a technique aimed at quantifying transient power and temperature variations in electronic systems under process uncertainty. The technique has been used to address the variability in the effective channel length, which is an area of particular importance. We have drawn a comparison with direct sampling, which has confirmed the efficiency of our technique in terms of accuracy and speed.

We have also presented a process-variation-aware technique for dynamic steady-state power and temperature analysis and have proposed a technique for reliability analysis that seamlessly accounts for the variability in process parameters and, in particular, for the impact of process variation on the operating temperature. In this case, we have compared our performance with that of direct sampling by considering the influence not only of the effective channel length but also of the gate oxide thickness. The results have shown the efficiency of our solution with respect to both accuracy and speed.

The low computational demand featured by the proposed framework implies that our uncertainty analysis can be readily performed inside design-space-exploration loops, including those targeted at energy and reliability optimization with temperature-related constraints under process variation. This important virtue has been demonstrated by considering an energy-driven probabilistic optimization procedure under temperature- and reliability-related constraints. It has been shown that temperature has to be treated as a stochastic quantity in order to make electronic systems reliable.

Finally, note that, even though our framework has been illustrated by considering the effective channel length, gate oxide thickness, and a number of specific quantities of interest, it can be easily applied to other problems that require process variation to be effectively and efficiently taken into account.

6 Analysis under Workload Uncertainty

Starting with this chapter, we move our focus from process uncertainty to workload uncertainty. The latter has its own idiosyncrasies and, therefore, poses its own challenges to the designer of electronic systems.

6.1 Introduction

Similarly to Chapter 5, we propose a design-time system-level framework for analysis of electronic systems that depend on uncertain parameters. In this chapter, however, the source of uncertainty being considered is workload. As is the case with sampling methods (see Section 1.5), our technique treats the system at hand as a "black box" and thus is straightforward to apply in practice, since no handcrafting is required, and existing code need not be changed. Hence, the quantities that the framework is able to address are diverse, including those that are concerned with timing-, power-, and temperature-related characteristics of applications running on heterogeneous platforms.

In contrast to Chapter 5 and sampling methods, the framework presented in this chapter explores and exploits the nature of the problem—that is, the way the quantity of interest depends upon the uncertain parameters—by exercising the aforementioned "black box" at an adaptively chosen set of points. The adaptivity that we leverage is hybrid [55]: it is sensitive to both global (on the level of individual stochastic dimensions [64]) and, more importantly, local (on the level of individual points [76]) variations. This means that the framework is able to benefit from any peculiarities that might be present in the stochastic space, the space of the uncertain parameters. The adaptivity is the primary feature of our technique, which we discuss and illustrate next.

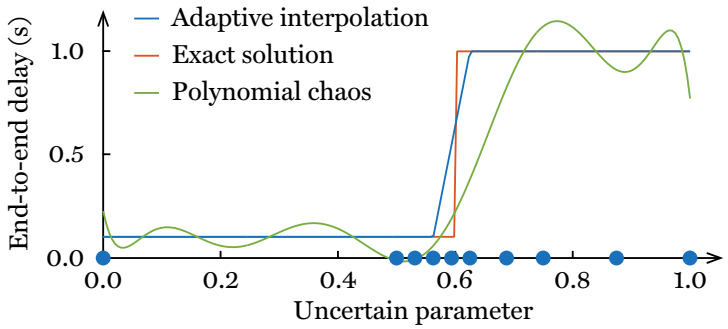

Figure 6.1: Example of the polynomial chaos decomposition and adaptive hierarchical interpolation applied to a nonsmooth quantity

6.2 Motivational Example

Due to its nature, the variability originating from process variation is typically smooth and well behaved. In such cases, uncertainty quantification based on polynomial chaos (PC) expansions [120] and other approximation techniques relying on global polynomials generally work well, as demonstrated in Chapter 5. On the other hand, the variability coming from sources such as workload often has steep gradients and favors nondifferentiability and even discontinuity. In such cases, PC expansions and similar techniques fail: they require an extremely large number of evaluations of the quantity of interest in order to deliver an acceptable level of accuracy and are consequently not worth it.

In order to illustrate this concern, let us consider an example. Suppose that our system has only one processing element, and it is running an application with only one task. Suppose also that the task has two branches and takes either one depending on the input data. Assume that one branch takes 0.1 s to execute and has probability 0.6, and the other branch takes 1 s and has probability 0.4. Our goal is to find the distribution of the end-to-end delay of the application. In this example, the quantity of interest is the end-to-end delay, and it coincides with the execution time of the task; hence, we already know the answer. Let us pretend we do not and try to obtain it by other means.

Suppose the above scenario is modeled by a uniformly distributed random variable $u : \Omega \to [0, 1]$. The execution time of the task (the end-to-end delay of the application) is 0.1 s if $u \in [0, 0.6]$, and it is 1 s if $u \in (0.6, 1]$. The response in this case is a step function, which is illustrated by the orange line in Figure 6.1.

First, we try to quantify the end-to-end delay by constructing and subsequently sampling a PC expansion founded on the Legendre polynomial basis [120]; see Appendix A.7. The green line in Figure 6.1 shows a ninth-level PC expansion ($l_c = l_q = 9$), which uses 10 points ($n_q = 10$). It can be seen that the approximation is poor—not to mention negative execution times—which

means that the follow-up sampling will also yield a poor approximation of the true distribution. The observed oscillating behavior is the well-known Gibbs phenomenon stemming from the discontinuity of the response. Regardless of the number of points spent, the oscillations will never go away completely.

Let us now see how the framework developed in this chapter solves the same problem. For the purpose of this experiment, our technique is constrained to make use of the same number of points as the PC expansion does. The result is the blue curve in Figure 6.1, and the adaptively chosen points are plotted on the horizontal axis. It can be seen that the approximation is good, and, in fact, it would be indistinguishable from the true response with a few additional points. Note that the adaptive procedure started to concentrate collocation nodes at the jump and paid little attention to the less interesting regions on either side of the jump. Having constructed such a representation, one can proceed to the calculation of the probability distribution of the quantity of interest, which, in general, should be done via sampling followed by such techniques as kernel density estimation [46]. The crucial point to note is that, analogously to the PC expansions leveraged in Chapter 5, this follow-up sampling does not involve the original system in any way, which implies that the operation costs practically nothing in terms of computation time.

The example described above illustrates the fact that the proposed framework is well suited for nonsmooth response surfaces. More generally, the adaptivity featured by our technique allows for a reduction in the costs associated with probabilistic analysis of the quantity under consideration as measured by the number of times the quantity needs to be evaluated in order to achieve a certain level of accuracy. The magnitude of reduction depends on the problem and can be substantial when the problem is well disposed to adaptation.

6.3 Problem Formulation

Consider an electronic system composed of two major components: a platform and an application. The platform is a collection of heterogeneous processing elements as defined in Section 2.1, whereas the application is a collection of interdependent tasks. The designer is interested in studying a quantity g that characterizes the system at hand from a certain perspective. Examples of g include the execution delay of the application or a specific task as well as the total energy consumption of the platform or a specific processing element.

The quantity of interest g depends on a set of parameters $\boldsymbol{u}$ that are uncertain at the design stage. Examples of $\boldsymbol{u}$ include the amount of data that the application must process, execution times of the tasks, and properties of the environment. The parameters $\boldsymbol{u}$ are given as a random vector $\boldsymbol{u} : \Omega \to \mathbb{R}^{n_u}$, which is defined on $(\Omega, \mathcal{F}, \mathbb{P})$ as in Appendix A.2, with an arbitrary but known distribution, whose cumulative distribution function (CDF) is denoted by F.

The dependency of g on $\boldsymbol{u}$ implies that g is random to the designer. For a given outcome of $\boldsymbol{u}$, however, the evaluation of g is assumed to be purely deterministic. This operation is traditionally performed by an adequate system simulator, and it is considered doable but computationally expensive.

Our objective in this chapter is to develop a framework for calculating the probability distribution of the quantity of interest g dependent on the uncertain parameters $\boldsymbol{u}$ so that this framework is able to efficiently handle nondifferentiable and potentially discontinuous dependencies between g and $\boldsymbol{u}$, which constitute an important class of problems for electronic-system design.

6.4 Previous Work

As noted in Section 1.5, a sampling method would be a reasonable solution to probabilistic analysis of electronic systems if these systems were computationally inexpensive to evaluate. In order to eliminate or reduce the costs associated with direct sampling, a number of techniques have been introduced.

We remark first that our work presented in Chapter 5 and thus the prior studies discussed in Section 5.4 are of relevance to this section. However, since those techniques are designed for a different source of uncertainty, namely process variation, we do not discuss them separately here; see also Section 6.2.

In the case of workload variation, timing analysis has attracted the most attention [88]. A seminal work on response time analysis of periodic tasks with random execution times on uniprocessor systems is reported in [32]. A novel analytical solution to this problem is presented in [105]; the solution makes milder assumptions and allows for addressing larger, previously unsolvable problems. The framework proposed in [96] makes use of real-time calculus in order to facilitate task scheduling by delivering probabilistic bounds on the resources provided to a task flow and the resources required by this task flow. In [98], the authors consider applications characterized by probabilities of execution and propose a heuristic that searches a design that has the least expected average power consumption. Independent periodic tasks with probabilistic execution times are analyzed in [124]; the work presents a scheme for optimistic reliability-aware power management targeted at energy minimization.

Studying the literature on probabilistic analysis of electronic systems related to this chapter (and Chapter 5 for that matter), one can note a pronounced trend: the generality and straightforwardness of sampling methods tend to be lost. The proposed techniques typically (a) require restrictive assumptions to be fulfilled, such as the absence of correlation; (b) are tailored to one concrete quantity of interest, such as the response time; and (c) require substantial effort in order to be deployed. However, one should keep in mind what is practical. First, although additional assumptions might make the mathematics analytically solvable, they often do not hold in reality and oversimplify the model. An exact analytical solution might also be extremely

complex, requiring a lot of computational resources upon evaluation. Furthermore, it is often the case that there is a robust simulator capable of calculating the quantity under consideration in the deterministic scenario. Switching to probabilistic analysis based on sophisticated techniques might mean discarding this battle-tested code and starting from scratch, which is wasteful.

Some of the techniques mentioned earlier, in fact, preserve the generality and straightforwardness of sampling methods. An example is our probabilistic framework presented in Chapter 5. This is because the construction of PC expansions in this framework is undertaken by means of nonintrusive spectral projections [120], which, similarly to sampling methods, do not need to look inside the "black box." However, as described in Section 6.2, nonsmoothness is a serious problem for global approximation based on polynomials. In particular, the convergence rate of PC expansions deteriorates substantially in such cases, requiring one to partition the stochastic space in order to alleviate the problem. Therefore, it is not straightforward to apply techniques such as the one in Chapter 5 to sources of uncertainty that exhibit nonsmoothness.

To conclude, the available techniques for probabilistic analysis of electronic systems are restricted in use. A flexible and easy-to-deploy framework capable of addressing nonsmooth uncertainty-quantification problems is needed.

6.5 Proposed Solution

The general solution strategy here is similar to the one outlined in Section 5.5. Recall first that making use of a sampling method is a compelling approach to uncertainty quantification. We would readily apply such a method to study the quantity of interest g if evaluating g had only a small cost, which, most of the time, it does not. Our solution to this quandary is to construct a lightweight representation of the heavy g and study this representation instead of g.

The surrogates that we build in this chapter are based on hierarchical interpolation with hybrid adaptivity, which is developed in [55, 64, 76]. In this case, g is evaluated at a number of strategically chosen collocation nodes, and any other values of g are reconstructed on demand, without involving g, using a set of basis functions that mediate between the collected values of g. The benefit of this approach is in the number of invocations of the quantity g: only a few evaluations of g are needed, and the rest of the analysis is powered by the constructed interpolant, which, in contrast to g, has a negligible cost.

The proposed framework is efficient at characterizing the impact of workload uncertainty and is straightforward to use in practice. The effectiveness of our approach is due to the aforementioned powerful approximation engine, which enables tackling diverse design problems while keeping the associated costs low. The usage of our approach is streamlined, since it has the same low entrance requirements as sampling techniques, which is also the case with our framework in Chapter 5: one only has to be able to evaluate the quantity of

interest given a particular outcome of the uncertain parameters. Moreover, it can be utilized in scenarios with limited knowledge of the joint probability distribution of the uncertain parameters, which are common in practice.

The solution process has four stages, which reflect the ones depicted in Figure 5.2. At Stage 1, the quantity of interest g and the uncertain parameters $\boldsymbol{u}$ are decided upon by the designer. At Stage 2, g is reparameterized in terms of an auxiliary random vector $\boldsymbol{z}$ extracted from $\boldsymbol{u}$, which is described in Section 6.6. At Stage 3, an interpolant of g is constructed by considering g as a deterministic function of $\boldsymbol{z}$ and evaluating g at a small set of carefully chosen points, which is detailed in Section 6.7. At Stage 4, the constructed interpolant of g is post-processed in order to calculate the desired statistics about g; in particular, the probability distribution of g is estimated by applying an arbitrary sampling method to the interpolant, which is discussed in Section 6.8.

The first stage of the framework is problem specific, and it will be exemplified in Section 6.9. In the following, we proceed directly to the second stage, which together with the third one should be approached with great care, since interpolation of multivariate functions is a challenging undertaking.

6.6 Probability Transformation

At Stage 2 of the framework, we change the parameterization of the problem from the random vector $\boldsymbol{u} : \Omega \to \mathbb{R}^{n_u}$ to an auxiliary random vector $\boldsymbol{z} : \Omega \to \mathbb{R}^{n_z}$ such that the support of the probability density function (PDF) of $\boldsymbol{z}$ is the unit hypercube $[0, 1]^{n_z}$, and that n_z has the smallest value required to retain the desired level of accuracy. The first task is standardization, which is done primarily for convenience. The second one is model order reduction, which identifies and eliminates excessive complexity and hence speeds up the subsequent solution process. The overall transformation is denoted by

$$\boldsymbol{u} = \mathbb{T}(\boldsymbol{z}) \tag{6.1}$$

where $\mathbb{T} : [0, 1]^{n_z} \to \mathbb{R}^{n_u}$. The quantity of interest g can now be computed as

$$g(\boldsymbol{u}) = (g \circ \mathbb{T})(\boldsymbol{z}) = g(\mathbb{T}(\boldsymbol{z})).$$

The attentive reader might already have a suitable candidate for $\mathbb{T}$: it is the one described in Appendix A.4, consolidated in Equation A.8, and utilized throughout Chapter 5 starting from Section 5.6.2. The way this transformation is applied in this chapter will be discussed further in Section 6.9.

6.7 Surrogate Construction

At Stage 3, an approximation of the quantity of interest is constructed using the interpolation algorithm presented in this section. The algorithm constitutes the core of the framework proposed in this chapter, and it features a

sparse structure, hierarchical construction, and hybrid adaptivity. The benefits of these characteristics are interconnected and can be summarized as follows: (a) the ability to efficiently address multidimensional problems, (b) the ability to progressively refine the approximation, and (c) the ability to perform this refinement strategically by virtue of fine-grained error control.

Hierarchical interpolation is introduced in Appendix A.6, and here we rely heavily on the results discussed in that section. The mathematics presented in Appendix A.6 and below are based on the development in [55, 64, 76].

Consider the quantity of interest g as a function of $\boldsymbol{z}$ via $\mathbb{T}$ as shown in Section 6.6. Assume that g belongs to $\mathcal{C}([0,1]^{n_z})$, the space of continuous functions on $[0,1]^{n_z}$; the assumption is not limiting in practice. As shown in Appendix A.6, g can be approximated by means of the following interpolant:

$$g \approx \mathcal{A}_{l_s}^{n_z}(g) = \mathcal{A}_{l_s-1}^{n_z}(g) + \sum_{\boldsymbol{i} \in \Delta\mathcal{I}_{l_s}^{n_z}} \sum_{\boldsymbol{j} \in \Delta\mathcal{J}_{\boldsymbol{i}}^{n_z}} \Delta(g \circ \mathbb{T})(\boldsymbol{x_{ij}}) e_{\boldsymbol{ij}} \tag{6.2}$$

where $\boldsymbol{i} \in \mathbb{N}_0^{n_z}$ is called a level index; $\boldsymbol{j} \in \mathbb{N}_0^{n_z}$ is called an order index; $\{\boldsymbol{x_{ij}}\}$ and $\{e_{\boldsymbol{ij}}\}$ are collocation nodes and basis functions, respectively; $\{\Delta(g \circ \mathbb{T})(\boldsymbol{x_{ij}})\}$ are hierarchical surpluses defined in Equation A.23; and $\Delta\mathcal{I}_{l_s}^{n_z}$ and $\Delta\mathcal{J}_{\boldsymbol{i}}^{n_z}$ are index sets defined in Equation A.18 and Equation A.22, respectively.

Due to the reasons clarified in Section 6.7.3, the interpretation of Equation 6.2 used in this chapter is different from the one used in Appendix A.6. Specifically, $l_s \in \mathbb{N}_0$ no longer represents an interpolation level but rather an interpolation step. Accordingly, $\mathcal{A}_{l_s}^{n_z}(g)$ represents the interpolant obtained by a certain interpolation step. In addition, all the index sets discussed here are generally subsets of their full-fledged counterparts defined in Appendix A.6.

Let us now turn to the choice of collocation nodes and basis functions.

6.7.1 Collocation Nodes

In order to gain computational efficiency, the integration grid should be fully nested, which is explained in Appendix A.6. Such a grid can be constructed using the family of Newton–Cotes rules [76]. In one dimension, a Newton–Cotes rule is a set of equidistant nodes on $[0,1]$. There are two types of Newton–Cotes rules: open and closed. The only difference between the two types is that the latter includes the endpoints (zero and one), whereas the former does not.

Technically, in order to be able to proceed to hierarchical interpolation, the chosen rules have to fulfill a certain condition, which is discussed in Appendix A.6 and can be seen in Equation A.20. Closed Newton–Cotes rules satisfy this condition, and they are the ones used in the original version of local adaptivity presented in [76]. The open rules, on the other hand, do not fulfill the condition close to the boundaries of the unit interval. However, according to our experience, open Newton–Cotes rules are a viable option, since they perform well in practice, which is also noted in [64]. In fact, we are able to obtain better results with open rules and, therefore, present them here.

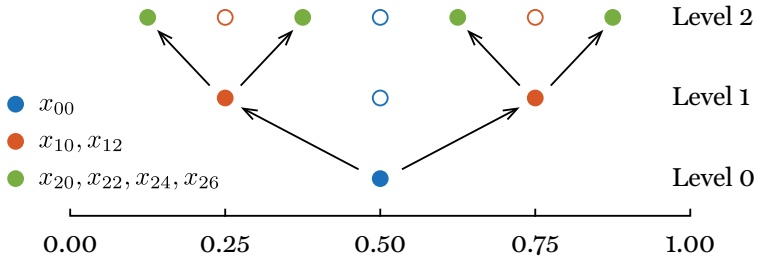

Figure 6.2: First three levels of the one-dimensional open Newton–Cotes grid with new nodes (*solid*) and inherited nodes (*hollow*)

The open Newton–Cotes rule of level $i \in \mathbb{N}_0$ is

$$\mathcal{X}_i^1 = \{x_{ij} : j \in \mathcal{J}_i^1\}$$

where

$$\begin{aligned} x_{ij} &= \frac{j+1}{n_i+1}, \\ \mathcal{J}_i^1 &= \{i-1\}_{i=1}^{n_i}, \text{ and} \\ n_i &= 2^{i+1}-1. \end{aligned}$$

Figure 6.2 depicts the first three levels of this rule. It can be seen that the number of nodes grows as 1, 3, 7, and so on, and that the rule is fully nested. In multiple dimensions, collocation nodes are formed as shown in Equation A.15.

6.7.2 Basis Functions

The basis functions that go hand in hand with open Newton–Cotes rules are piecewise linear functions. For $i = 0$ and $j = 0$,

$$e_{00}(x) = 1.$$

For $i > 0$ and $j = 0$ (close to the left endpoint),

$$e_{i0}(x) = \begin{cases} 2 - (n_i+1)\,x, & \text{if } x < \frac{2}{n_i+1}; \\ 0, & \text{otherwise.} \end{cases}$$

For $i > 0$ and $j = n_i - 1$ (close to the right endpoint),

$$e_{i,n_i-1}(x) = \begin{cases} (n_i+1)\,x - n_i + 1, & \text{if } x > \frac{n_i-1}{n_i+1}; \\ 0, & \text{otherwise.} \end{cases}$$

In other cases,

$$e_{ij}(x) = \begin{cases} 1 - (n_i+1)\,|x - x_{ij}|, & \text{if } |x - x_{ij}| < \frac{1}{n_i+1}; \\ 0, & \text{otherwise.} \end{cases}$$

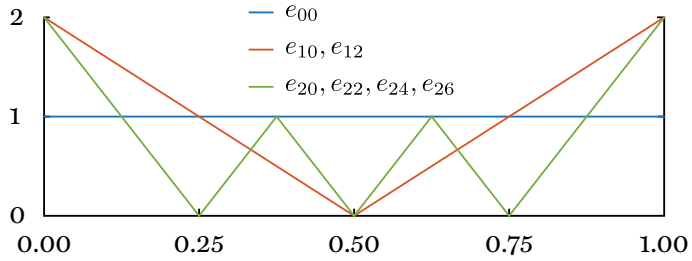

Figure 6.3: First three levels of the one-dimensional piecewise linear basis

The basis functions that correspond to the first three levels of one-dimensional interpolation are depicted in Figure 6.3. In multiple dimensions, basis functions are formed as shown in Equation A.15, which results in

$$e_{\boldsymbol{ij}}(\boldsymbol{x}) = \prod_{k=1}^{n} e_{i_k j_k}(x_k).$$

Additionally, let us calculate the volumes (the integrals over the whole domain) of the aforementioned piecewise linear functions; these volumes are required in the continuation. For $i = 0$ and $j = 0$,

$$w_{00} = 1.$$

For $i > 0$ and $j \in \{0, n_i - 1\}$,

$$w_{ij} = \frac{2}{n_i + 1}.$$

In other cases,

$$w_{ij} = \frac{1}{n_i + 1}.$$

The volumes of multidimensional basis functions are products of the corresponding one-dimensional volumes as follows:

$$w_{\boldsymbol{ij}} = \prod_{k=1}^{n_z} w_{i_k j_k}. \tag{6.3}$$

Now, imagine a function that is nearly flat on the first half of $[0, 1]$ and rather irregular on the other. Under these circumstances, it is natural to expect that, in order to attain the same accuracy, the first half should require many fewer collocation nodes than the other one; recall the example given in Figure 6.1. However, if we followed the usual construction procedure described in Appendix A.6, we would not be able to benefit from this idiosyncrasy. Both sides would be treated equally, and all the nodes of each interpolation level would be added to the interpolant, which is wasteful. The solution to this problem is to make the interpolation algorithm adaptive, which we discuss next.

6.7.3 Hybrid Adaptivity

In order to make the algorithm adaptive, we first need to decide on a criterion used for measuring the accuracy of the interpolant $\mathcal{A}_{l_s}^{n_z}(g)$ in Equation 6.2 at any point in $[0,1]^{n_z}$. Then, when refining the interpolant, instead of evaluating the quantity of interest g at all possible nodes, we choose only those that are located in the regions with poor accuracy as indicated by the criterion.

We already have a good foundation for building the above-mentioned criterion. Hierarchical surpluses, which are introduced in Appendix A.6 and defined in Equation A.23, are natural indicators of the interpolation error: they are the difference between the true values of g and those estimated by $\mathcal{A}_{l_s}^{n_z}(g)$ at the nodes of the underlying integration grid. Therefore, hierarchical surpluses can be recycled in order to effectively identify problematic regions.

We proceed as follows. First, a score is assigned to each node $\boldsymbol{x_{ij}}$ or, equivalently, to each pair of a level index $\boldsymbol{i}$ and an order index $\boldsymbol{j}$ as follows:

$$s_{\boldsymbol{ij}} = |\Delta(g \circ \mathbb{T})(\boldsymbol{x_{ij}}) w_{\boldsymbol{ij}}| \tag{6.4}$$

where $\Delta(g \circ \mathbb{T})(\boldsymbol{x_{ij}})$ is the surplus at the node as defined in Equation A.23, and $w_{\boldsymbol{ij}}$ is the volume of the corresponding basis function as shown in Equation 6.3. The above score is utilized for guiding the algorithm as explained below.

Each hierarchical interpolant $\mathcal{A}_{l_s}^{n_z}$ is characterized by a set of level indices $\mathcal{I}_{l_s}^{n_z}$, and each level index $\boldsymbol{i} \in \mathcal{I}_{l_s}^{n_z}$ by a set of order indices $\Delta\mathcal{J}_{\boldsymbol{i}}^{n_z}$. At each interpolation step $l_s \in \mathbb{N}_0$, a single level index

$$\boldsymbol{i}_{l_s} \in \mathcal{I}_{l_s-1}^{n_z}$$

is chosen where $\mathcal{I}_{-1}^{n_z} = \{\mathbf{0}\}$. This index gives birth to $\Delta\mathcal{I}_{l_s}^{n_z}$ and $\{\Delta\mathcal{J}_{\boldsymbol{i}}^{n_z} : \boldsymbol{i} \in \Delta\mathcal{I}_{l_s}^{n_z}\}$, forming the increment given on the right-hand side of Equation 6.2.

The level index set $\Delta\mathcal{I}_{l_s}^{n_z}$ contains so-called admissible forward neighbors of the chosen $\boldsymbol{i}_{l_s}$. The forward neighbors of an index $\boldsymbol{i}$ are given by

$$\{\boldsymbol{i} + \mathbf{1}_k\}_{k=1}^{n_z}$$

where $\mathbf{1}_k \in \{0,1\}^{n_z}$ is a vector whose elements are zero except for element k, which is equal to unity. An index $\boldsymbol{i}$ is called admissible if its inclusion into the index set $\mathcal{I}_{l_s}^{n_z}$ under consideration keeps the set admissible. Finally, $\mathcal{I}_{l_s}^{n_z}$ is admissible if it satisfies the following condition [64]:

$$\boldsymbol{i} - \mathbf{1}_k \in \mathcal{I}_{l_s}^{n_z}$$

for $\boldsymbol{i} \in \mathcal{I}_{l_s}^{n_z}$ and $k \in \{1, \ldots, n_z\}$ where the cases with $i_k = 0$ need no check.

Let us now turn to the content of $\Delta\mathcal{J}_{\boldsymbol{i}}^{n_z}$ where $\boldsymbol{i} = \boldsymbol{i}_{l_s} + \mathbf{1}_k$ for some k. It also contains admissible forward neighbors; however, they are order indices, and their construction is different from the one used in the case of $\Delta\mathcal{I}_{l_s}^{n_z}$. These indices are identified by inspecting the backward neighborhood of $\boldsymbol{i}$, which is

analogous to the forward one but in the other direction. For each backward neighbor $\boldsymbol{i} - \mathbf{1}_m$ and each $\boldsymbol{j} \in \Delta\mathcal{J}^{n_z}_{\boldsymbol{i}-\mathbf{1}_m}$, we first check the condition

$$s_{\boldsymbol{i}-\mathbf{1}_m,\boldsymbol{j}} \geq \epsilon_s$$

where ϵ_s is a user-defined constant referred to as the score error. If the condition holds, the forward neighbors of $\boldsymbol{j}$ in dimension k are added to $\Delta\mathcal{J}^{n_z}_{\boldsymbol{i}}$. This procedure is illustrated in Figure 6.2 for open Newton–Cotes rules in one dimension. The arrows emerging from a collocation node connect the node with its forward neighbors. In general, each node has two forward neighbors in each dimension. The order indices of these nodes are

$$\begin{aligned}&(j_1, \ldots, 2j_k \quad\;\; , \ldots, j_{n_z}) \text{ and}\\ &(j_1, \ldots, 2j_k + 2, \ldots, j_{n_z}).\end{aligned}$$

The above refinement procedure should be performed for each level index $\boldsymbol{i} \in \Delta\mathcal{I}^{n_z}_{l_s}$ with respect to each dimension $k \in \{1, \ldots, n_z\}$.

The choice of $\boldsymbol{i}_{l_s} \in \mathcal{I}^{n_z}_{l_s-1}$ at each step l_s in Equation 6.2 is made as follows. First, each index can be picked at most once. The rest is resolved by prioritizing the candidates. It is reasonable to compute the priority of a level index $\boldsymbol{i}$ based on the scores of the order indices associated with this level index, that is, based on the scores of $\mathcal{J}^{n_z}_{\boldsymbol{i}}$. We set the priority to the average score

$$s_{\boldsymbol{i}} = \frac{1}{\#\Delta\mathcal{J}^{n_z}_{\boldsymbol{i}}} \sum_{\boldsymbol{j} \in \Delta\mathcal{J}^{n_z}_{\boldsymbol{i}}} s_{\boldsymbol{ij}}.$$

Thus, at each step l_s, the index $\boldsymbol{i}$ with the highest $s_{\boldsymbol{i}}$ is promoted to $\boldsymbol{i}_{l_s}$.

The final question to answer is the stopping condition of the approximation process in Equation 6.2. Apart from the natural constraints on the maximum number of function evaluations and the maximum interpolation level (the original l_s in Appendix A.6), we rely on the following criterion. Given two additional user-defined constants ϵ_a and ϵ_r, which are referred to as the absolute and relative errors, respectively, the process is terminated as soon as

$$\max_{\boldsymbol{ij}} |\Delta(g \circ \mathbb{T})(\boldsymbol{x}_{\boldsymbol{ij}})| \leq \max\{\epsilon_a, \epsilon_r(g_{\max} - g_{\min})\} \tag{6.5}$$

where $g_{\min}$ and $g_{\max}$ are the minimum and maximum observed values of g, respectively. The left-hand side of Equation 6.5 corresponds to the largest surplus whose level index has not been refined yet, that is, has not been considered as $\boldsymbol{i}_{l_s}$ at some step l_s. The above criterion is an adequate technique for curtailing the interpolation process, since it is based on the actual progress.

The adaptivity presented in this subsection is referred to as hybrid, since it combines features of global and local adaptivity [55]. Local adaptivity operates on the level of individual nodes [76] and is discussed in Section 6.2. By contrast, global adaptivity operates on the level of individual dimensions [64].

The intuition behind global adaptivity is that, in general, the input variables manifest themselves (impact g) differently, and the interpolation algorithm is likely to benefit by prioritizing those variables that are the most influential.

So far, we have formalized an efficient algorithm for adaptive hierarchical interpolation in multiple dimensions. The main equation is Equation 6.2 where $\Delta(g \circ \mathbb{T})(\boldsymbol{x_{ij}})$, $\boldsymbol{x_{ij}}$, and $e_{\boldsymbol{ij}}$ are the ones in Equation A.23, Section 6.7.1, and Section 6.7.2, respectively; and the interpolation is undertaken according to the rules in Section 6.7.3. Next, we discuss a few implementation details.

6.7.4 Implementation

The life cycle of interpolation has roughly two stages: construction and usage. The construction stage invokes g at a set of collocation nodes and produces certain artifacts. The usage stage estimates the values of g at a set of arbitrary points by manipulating these artifacts. In this subsection, we provide the pseudocode for the two stages in order to give a better sense of the technique.

Let us first make a general note. According to our experience, it is beneficial to the clarity and ease of implementation to collapse the two sums in Equation 6.2 into one. This requires storing a level index $\boldsymbol{i} = (i_k) \in \mathbb{N}_0^{n_z}$ and an order index $\boldsymbol{j} = (j_k) \in \mathbb{N}_0^{n_z}$ for each node. It is also advantageous to encode each pair (i_k, j_k) as a single unsigned integer, which, in particular, eliminates excessive memory usage. In multiple dimensions, this results in a vector $\boldsymbol{\iota} = (\iota_k) \in \mathbb{N}_0^{n_z}$, which we simply call an index. Our encoding is

$$\iota_k = i_k \vee (j_k \ll n_{\text{bits}})$$

where $\vee$ and $\ll$ stand for the bitwise OR and logical left shift, respectively, and n_{bits} is the number of bits reserved for storing level indices, which can be adjusted according to the maximum permitted depth of interpolation.

The pseudocode for the construction stage is given in Algorithm 6.1. The input is a subroutine called Algorithm G that evaluates $g \circ \mathbb{T}$. The output is a structure `interpolant` that contains the artifacts of the interpolation. These artifacts are a set of tuples $\{(\boldsymbol{\iota}_k, \Delta g(\boldsymbol{x}_{\boldsymbol{\iota}_k}))\}$, which is a comprehensive description of a hierarchical interpolant. The pseudocode works as follows.

Line 1: Each iteration of the loop corresponds to an interpolation step l_s in Equation 6.2. The progress is captured by a structure `state`. The `strategy` object represents the adaptation strategy utilized, and it operates in accordance with Section 6.7.3. The `Continue?` method of `strategy` checks if any of the stopping conditions is satisfied, in which case the process is terminated.

Line 2: The `Next` method of `strategy` consumes the previous state and returns the initial state of the ongoing interpolation step. In particular, it populates the `indices` field of `state` with the indices of the step. The rest of the loop's body populates the rest of `state`'s fields so that `strategy` can adequately execute its functionality at the beginning of the next iteration.

Algorithm 6.1: Construction of an adaptive hierarchical interpolant

Input: Algorithm G // a subroutine evaluating $g \circ \mathbb{T}$
Output: interpolant
1: **while** strategy.Continue?(state) **do**
2: state ← strategy.Next(state)
3: state.nodes ← grid.Compute(state.indices)
4: state.values ← Algorithm G(state.nodes)
5: state.estimates ← Algorithm 6.2(interpolant, state.nodes)
6: state.surpluses ← Subtract(state.values, state.estimates)
7: state.scores ← strategy.Score(state.surpluses)
8: interpolant.Append(state.indices, state.surpluses)
9: **end while**
10: **return** interpolant

Line 3: The `grid` object represents the interpolation grid that is being utilized, and its `Compute` method calculates the collocation nodes $\{\boldsymbol{x}_{\boldsymbol{\iota}_k}\}$ that correspond to the given indices $\{\boldsymbol{\iota}_k\}$; see Section 6.7.1.

Line 4: Algorithm G evaluates $g \circ \mathbb{T}$ at the collocation nodes. This is by far the most time consuming operation of the algorithm, as g is generally expensive to execute. This operation is also a prominent candidate for parallelization, since the algorithm does not impose any particular evaluation order.

Line 5: Algorithm 6.2 is a subroutine that exercises the interpolant constructed so far at the collocation nodes and thereby approximates the values obtained on line 4. This algorithm will be discussed separately later on.

Line 6: The `Subtract` subroutine computes the difference between the actual and approximated values of g, which yields the hierarchical surpluses $\{\Delta(g \circ \mathbb{T})(\boldsymbol{x}_{\boldsymbol{\iota}_k})\}$ (see Equation A.23) of the current interpolation step.

Line 7: The `Score` method of `strategy` calculates the scores of the collocation nodes based on their surpluses as described in Section 6.7.3.

Line 8: The `Append` method of `interpolant` refines the interpolant by extending it with the indices and surpluses of the completed iteration.

We now turn our attention to the usage stage of an interpolant. The pseudocode is given in Algorithm 6.2. This subroutine is also used in Algorithm 6.1 on line 5. Let us make a couple of observations regarding this subroutine.

Line 3: The inner loop corresponds to an unfolded version of Equation 6.2; that is, there is no separation into the individual interpolation steps taken.

Line 4: The `basis` object represents the interpolation basis that is being used, and its `Compute` method evaluates the basis functions $\{e_{\boldsymbol{\iota}_k}\}$ that correspond to the given indices $\{\boldsymbol{\iota}_k\}$ at arbitrary points; see Section 6.7.2.

It is worth noting that the `strategy`, `grid`, and `basis` objects conform to certain interfaces and can be easily swapped out. This makes the two algo-

Algorithm 6.2: Evaluation of an adaptive hierarchical interpolant

```
Input: interpolant, points
Output: estimates                                   // approximated values
for point in points do
    estimate ← 0
    for (index, surplus) in interpolant do
        weight ← basis.Compute(index, point)
        estimate ← estimate + surplus × weight
    end for
    estimates.Append(estimate)
end for
return estimates
```

rithms very general and reusable with different configurations. In particular, the adaptation strategy can be fine-tuned for each particular problem.

To recapitulate, in this section, the approximation engine of our framework for probabilistic analysis of electronic systems under workload variation has been presented. It is consolidated in Algorithm 6.1 and Algorithm 6.2.

6.8 Post-Processing

At Stage 4 of the proposed framework (see Section 6.5), the constructed interpolant $\mathcal{A}_{l_s}^{n_z}(g)$ of the quantity of interest g is processed according to the designer's requirements. The post-processing in this chapter is similar to that described in Section 5.6.4. Namely, probabilistic analysis of g is carried out solely on $\mathcal{A}_{l_s}^{n_z}(g)$, and g is never invoked again. Since $\mathcal{A}_{l_s}^{n_z}(g)$ is a lightweight representation of g, this analysis has a negligible computational cost.

The distribution of g can be efficiently estimated via a sampling method of choice. In this case, one draws independent samples from the distribution of $\boldsymbol{z}$ and evaluates $\mathcal{A}_{l_s}^{n_z}(g)$ at those points in accordance with Algorithm 6.2. Having collected a large enough set of samples, the distribution of g can be computed by employing such techniques as kernel density estimation [46]. Probabilities of various events can be straightforwardly estimated as well.

In addition, the expectation and variance of g can be computed without sampling. Using a carefully chosen transformation $\mathbb{T}$ (recall Section 6.6), g can be reparameterized in terms of independent variables that are uniformly distributed on $[0,1]^{n_z}$; this transformation will be discussed further in Section 6.9.2. In this scenario, the density function of $\boldsymbol{z}$ equals unity. Therefore, using Equation 6.2, we obtain the following analytical expression:

$$\mathbb{E}g \approx \mathbb{E}\mathcal{A}_{l_s}^{n_z}(g) = \int_{[0,1]^{n_z}} \mathcal{A}_{l_s}^{n_z}(g)(\boldsymbol{z})\mathrm{d}\boldsymbol{z} = \sum_{\boldsymbol{i}\in\mathcal{I}_{l_s}^{n_z}} \sum_{\boldsymbol{j}\in\Delta\mathcal{J}_{\boldsymbol{i}}^{n_z}} \Delta(g\circ\mathbb{T})(\boldsymbol{x_{ij}})w_{\boldsymbol{ij}}$$

where w_{ij} is the volume of a basis function as shown in Equation 6.3.

Regarding the variance, observe in Equation A.2 that $\mathbb{V}\mathrm{ar}(g)$ can be assembled from two components: the expectation of g, which we already have, and the expectation of g^2, which is missing. The solution is to let $h = (g, g^2)$ be the quantity of interest instead of g. The expectations of both g and g^2 then become available analytically, and $\mathbb{V}\mathrm{ar}(g)$ can be computed using Equation A.2. This approach can be generalized to probabilistic moments of higher orders.

The careful reader might have noted a problem with the calculation of $\mathbb{V}\mathrm{ar}(g)$ presented above: h is a vector, but g has been depicted so far as having a one-dimensional codomain. However, this has been done for clarity; our framework does not have such a limitation, as explained in Remark 6.1.

Remark 6.1. *The mathematics and pseudocode remain unchanged for vector-valued quantities. The only difference is that, since hierarchical surpluses naturally inherit the output dimensionality of g, the operations that involve them should be adequately adjusted. If the outputs are on different scales or have different accuracy requirements, one might consider having different ϵ_a and ϵ_r in Equation 6.5 for different outputs. In that case, one also has to revisit Equation 6.4 and devise a more sensible strategy for scoring collocation nodes, such as rescaling individual outputs and then calculating $\|\cdot\|_2$ or $\|\cdot\|_\infty$.*

To summarize, once an interpolant of g has been constructed, the distribution of g is estimated using a sampling method that is applied to the interpolant. The proposed framework naturally extends to quantities with multiple outputs, and it provides analytical formulae for the expectation and variance.

Lastly, let us recall that the evaluation of the quantity of interest is an expensive operation. The proposed technique is designed to keep this expense as low as possible by choosing evaluation points adaptively, which is unlike traditional sampling methods. Moreover, in contrast to PC expansions and similar techniques, our framework is well suited for nonsmooth response surfaces.

6.9 Illustrative Application

The agenda for this section is as follows. First, we exemplify Stage 1 of our framework by introducing a number of quantities of interest g, which illustrate the broad applicability of the framework and thereby give the reader a better sense of its utility. Second, we turn to Stage 2 and highlight a transformation $\mathbb{T}$ that is the appropriate one to use in the majority of cases. Third, we proceed directly to Stage 4 and illustrate a potential output of the framework.

6.9.1 Problem Formulation

Assume the system, power, and temperature models described in Section 2.1, Section 2.2, Section 2.3, respectively. Assume also the application model utilized in Section 3.6.2 except for the requirement about being periodic.

Let us first touch upon the timing aspects of the system in question. Each of the n_t tasks of the application has a start time and a finish time, which are denoted by b_i and d_i, respectively. Let also $\boldsymbol{b} = (b_i)_{i=1}^{n_t}$ and $\boldsymbol{d} = (d_i)_{i=1}^{n_t}$. Other timing characteristics can then be derived from $\boldsymbol{b}$ and $\boldsymbol{d}$. For example, the end-to-end delay of the application, which is the difference between the finish time of the latest task and the start time of the earliest task, is as follows:

$$\text{End-to-end delay} = \max_{i=1}^{n_t} d_i - \min_{i=1}^{n_t} b_i. \tag{6.6}$$

Suppose now that the execution times of the tasks depend on the uncertain parameters $\boldsymbol{u}$ introduced in Section 6.3. Then $\boldsymbol{b}$ and $\boldsymbol{d}$ depend on $\boldsymbol{u}$. Hence, the end-to-end delay given in Equation 6.6 does too, and it constitutes a quantity g that the designer might be interested in analyzing. Note that, since the $\min$ and $\max$ functions are nondifferentiable, the same is true for g. Therefore, g is nonsmooth, which renders the PC decomposition and similar techniques inappropriate in this case in general, which is discussed in Section 6.2.

Remark 6.2. *The behavior of g with respect to continuity, differentiability, and smoothness cannot generally be inferred from the behavior of $\boldsymbol{u}$. Even when the parameters behave perfectly, g might still exhibit nondifferentiability or even discontinuity, which depends on how g functions internally. For example, as shown in [105], even if the execution times of tasks are continuous, end-to-end delays are very often discontinuous due to the actual scheduling policy.*

Let us move on to power. The total energy consumed by the n_p processing elements during an application run can be estimated using a power profile $\mathbf{P}$, which is defined in Equation 2.1, as follows:

$$\text{Total energy} = \sum_{i=1}^{n_p} \int p_i(t)\mathrm{d}t \approx \Delta t \left\|\mathbf{P}\right\|_1 \tag{6.7}$$

where p_i denotes the power consumption of processing element i, Δt is the sampling interval, and $\|\cdot\|_1$ stands for the Manhattan norm. Since $\boldsymbol{b}$ and $\boldsymbol{d}$ depend on $\boldsymbol{u}$, the power consumption of the system is also dependent on $\boldsymbol{u}$. Consequently, the total energy given in Equation 6.7 depends on $\boldsymbol{u}$ and is a candidate for g. Remark 6.2 applies in this context to the full extent.

Let us now turn to temperature. The maximum temperature that the platform reaches during an application run can be estimated using a temperature profile $\mathbf{Q}$, which is defined in Equation 2.2, as follows:

$$\text{Maximum temperature} = \max_{i=1}^{n_p} \sup_t q_i(t) \approx \left\|\mathbf{Q}\right\|_\infty \tag{6.8}$$

where q_i denotes the heat dissipation of processing element i, and $\|\cdot\|_\infty$ stands for the uniform norm. Since the power consumption of the platform is affected by $\boldsymbol{u}$, the corresponding heat dissipation is influenced by $\boldsymbol{u}$ as well. Therefore,

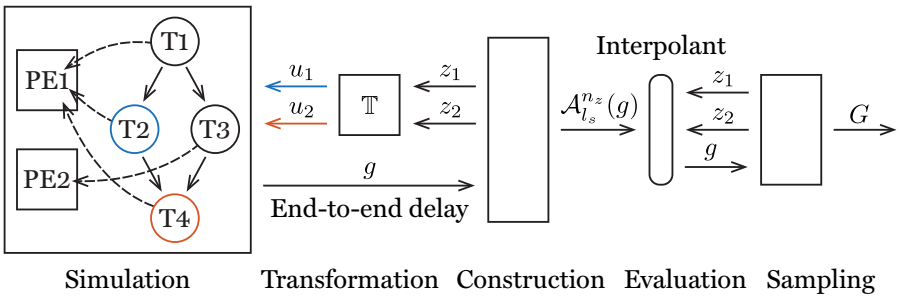

Figure 6.4: Proposed solution applied to the end-to-end delay of an application in which two out of four tasks have uncertain execution times

the maximum temperature in Equation 6.8 is also a potential quantity of interest g. Note that, due to the maximization involved in the calculations, the quantity is nondifferentiable and hence cannot generally be adequately addressed using polynomial approximations; recall also the concern in Remark 6.2.

To summarize, we have covered three aspects of electronic systems, namely timing, power, and temperature, and introduced a number of quantities that the designer is typically interested in analyzing. These quantities will be discussed further in the section on experimental results, Section 6.10.

Let us employ one of the introduced quantities in order to have a concrete example to work with in this section. The problem being addressed is depicted on the left-hand side of Figure 6.4. We consider a heterogeneous platform with two processing elements denoted by PE1 and PE2 and an application with four tasks denoted by T1–T4; the setup will be described in detail in Section 6.10. The data dependencies between the tasks and their mapping onto the processing elements can be seen in Figure 6.4. The quantity of interest g is the application's end-to-end delay defined in Equation 6.6. The uncertain parameters $\boldsymbol{u}$ are the execution times of T2 and T4 denoted by u_1 and u_2, respectively.

The large rectangle on the left-hand side of Figure 6.4 is a "black box" that evaluates g given $\boldsymbol{u}$. It takes an assignment of the execution times u_1 and u_2 and outputs the calculated end-to-end delay g. In practice, this evaluation often involves a system simulator, such as Sniper [10], in which case the modeling capabilities of this simulator are naturally inherited by our technique.

Targeting the practical scenario described in Section 5.6.1, the marginal distributions and correlation matrix of $\boldsymbol{u}$ are assumed to be available. Without loss of generality, each marginal distribution is a four-parameter beta distribution shown in Equation 5.12. Furthermore, the execution times are assumed to be correlated based on the dependencies between the corresponding tasks as defined by the structure of the application's task graph. Specifically, the closer task i and task j are in the graph as measured by the number of edges between vertex i and vertex j, the more strongly u_i and u_j are correlated.

6.9.2 Probability Transformation

At Stage 2 of our workflow outlined in Section 6.5, a suitable transformation $\mathbb{T}$, which is required in Equation 6.1, is chosen. We utilize the one shown in Equation A.8 and reduce $\boldsymbol{u} : \Omega \to \mathbb{R}^{n_u}$ to $\boldsymbol{z} : \Omega \to [0,1]^{n_z}$ so that the latter is uniformly distributed. In this case, the rightmost $\mathbb{T}_1$ in Equation A.8 is simplified, since the marginals of $\boldsymbol{z}$ are already uniform. Note that the model-order-reduction functionality of the chosen $\mathbb{T}$ is engaged; it eliminates redundant stochastic dimensions and, therefore, assists the subsequent interpolation.

The result is that the obtained vector $\boldsymbol{z}$ conforms to the requirements listed in Section 6.6: the codomain of $\boldsymbol{z}$ is $[0,1]^{n_z}$, and it has the smallest number of dimensions that are necessary in order to preserve a certain level of accuracy.

In Figure 6.4, $\mathbb{T}$ is depicted as a small square. In this particular example, the stochastic dimensionality is found to be the same before and after $\mathbb{T}$, which is indicated by the two incoming and two outgoing arrows. The depicted component takes an assignment of the auxiliary variables z_1 and z_2 and outputs the execution times u_1 and u_2 in accordance with their joint distribution.

In the following, we proceed directly to Stage 4, since Stage 3, which is given in the middle of Figure 6.4, is standard: using a number of strategic invocations of the simulator of g (the "black box"), it delivers a lightweight surrogate $\mathcal{A}_{l_s}^{n_z}(g)$, which is illustrated by a rounded rectangle in Figure 6.4.

6.9.3 Post-Processing

Having constructed the interpolant $\mathcal{A}_{l_s}^{n_z}(g)$, the designer starts to work solely with this interpolant, which corresponds to Stage 4 of our framework.

Suppose the designer is interested in the probability distribution of g. In this scenario, $\mathcal{A}_{l_s}^{n_z}(g)$ should be sampled, which is represented by the rightmost box in Figure 6.4. The operation corresponds roughly to Algorithm 6.2: the interpolant receives z_1 and z_2 and returns an approximation of the value of g at that point. Recall that the computational cost of this sampling is negligible, since g is not involved. The collected samples, which are denoted by G in Figure 6.4, are then utilized in order to estimate the distribution of g.

Figure 6.5 depicts the result. The blue line shows the PDF of g computed by applying kernel density estimation [46] to the data set G. The orange line, which is barely visible behind the blue line, shows the actual density of g; its calculation is explained in Section 6.10. It can be seen that our solution closely matches the exact one. In addition, the green line illustrates the estimate that the designer would obtain if g was sampled directly using the same number of evaluations as the one consumed by the proposed framework. It can be seen that, given the same budget, the solution delivered by our technique is substantially closer to the exact one than the one delivered by direct sampling.

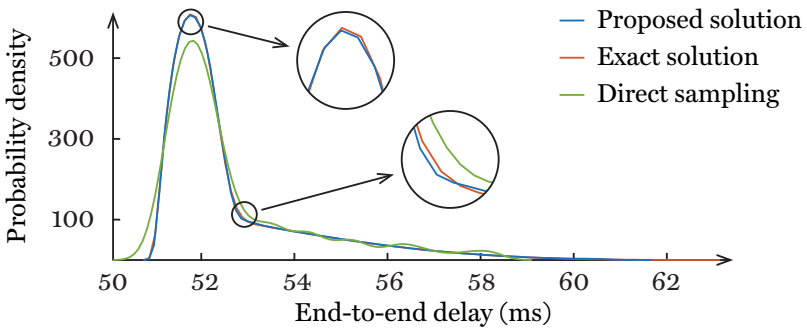

Figure 6.5: Example of probability density functions computed using the proposed solution and direct sampling

6.10 Experimental Results

In this section, we assess the proposed framework. All the experiments are conducted on a GNU/Linux machine equipped with 16 Intel Xeon E5520 2.27 GHz processors and 24 GB of RAM. All the source code, configuration files, and input data used in the experiments are available online at [20].

We consider three platform sizes n_p, two application sizes n_t, and three quantities of interest g. Specifically, n_p is 2, 4, or 8; n_t is 10 or 20; and g is the end-to-end delay, total energy consumption, or maximum temperature. The three quantities of interest are defined in Equation 6.6, Equation 6.7, and Equation 6.8, respectively. Therefore, we address 18 problems in total, which correspond to different combinations of n_p, n_t, and g. At this point, it might be helpful to recall the illustrative application depicted in Figure 6.4.

Each problem is configured as follows. A platform with n_p processing elements and an application with n_t tasks are randomly generated by TGFF [34]. The tool generates a table for each processing element that specifies certain properties of the tasks when they are mapped to that processing element. Namely, each table assigns two numbers to each task: a reference execution time and a power consumption value, which are chosen uniformly between 10 and 50 ms and between 5 and 25 W, respectively. The application is scheduled using a list scheduler [1]. The mapping of the application is fixed and obtained by scheduling the tasks based on their reference execution times and assigning them to the earliest available processing elements (a shared ready list).

Similarly to the previous chapters, the construction of thermal RC circuits, which are required for temperature analysis, is delegated to HotSpot [100]. The floorplan of each platform is a regular grid where each processing element occupies 4 mm^2. The modeling of the static component of power consumption is based on a linear fit to a set of SPICE simulations of a series of CMOS invertors. The sampling interval Δt of power and temperature profiles is 1 ms.

The uncertain parameters $\boldsymbol{u}$ are the execution times of the tasks, and their marginal distributions and correlation matrix are as described in Section 6.9.1; all other parameters are deterministic. Regarding the marginal distributions, which are beta distributions as shown in Equation 5.12, the left c and right d endpoints of their supports are set to 80% and 120%, respectively, of the reference execution times generated by TGFF as described earlier. The parameters a and b are set to two and five, respectively, for all tasks, which skews the distributions toward the left endpoints. The model-order-reduction parameter η in Equation A.7 is set to 0.9, which results in $n_z = 2$ and $n_z = 3$ independent variables for applications with $n_t = 10$ and $n_t = 20$ tasks, respectively.

The configuration of the interpolation algorithm—including the collocation nodes, basis functions, and adaptation strategy with stopping conditions—closely follows the description given in Section 6.7. The parameters ϵ_a, ϵ_r, and ϵ_s are set to around 10^3, 10^2, and 10^4, respectively, depending on the problem.

The performance of our framework with respect to each problem is assessed as follows. First, as in Chapter 5, the true probability distribution of g is estimated by sampling g directly and extensively. Direct sampling means that there is no intermediate representation or model order reduction involved. Second, we construct an interpolant of g and estimate the distribution of g by sampling this interpolant, which is discussed in Section 6.7 and Section 6.8. In both cases, we draw 10^5 samples; recall, however, that, unlike the cost of direct sampling, the cost of sampling the interpolant is negligible. Third, we perform another round of direct sampling. This time, the number of samples taken is equal to the number of times g is invoked during the interpolation process.

In each of the aforementioned three cases, sampling is performed in accordance with a Sobol sequence, which is a low-discrepancy sequence featuring better convergence properties than those of classical Monte Carlo (MC) sampling [57]. As a result, we obtain three estimates of the probability distribution of the quantity of interest: reference (the one considered to be exact), proposed (the one based on our interpolation), and direct (the one equal in terms of the number of evaluations of g to the proposed solution). The last two are compared with the first one. For computing the proximity between two distributions, we use the well-known Kolmogorov–Smirnov (KS) statistic [90], which is the supremum over the distance (pointwise) between two empirical distribution functions and is consequently a stringent error indicator.

6.10.1 Approximation Accuracy

The results are given in Figure 6.6, Figure 6.7, and Figure 6.8, which correspond to the end-to-end delay, total energy, and maximum temperature, respectively. Each figure contains six plots arranged in a three-by-two grid. The three rows of the grid correspond to the three platform sizes, and the two columns to the two application sizes. The horizontal axis of each plot shows the number of evaluations of the quantity of interest g, and the vertical one

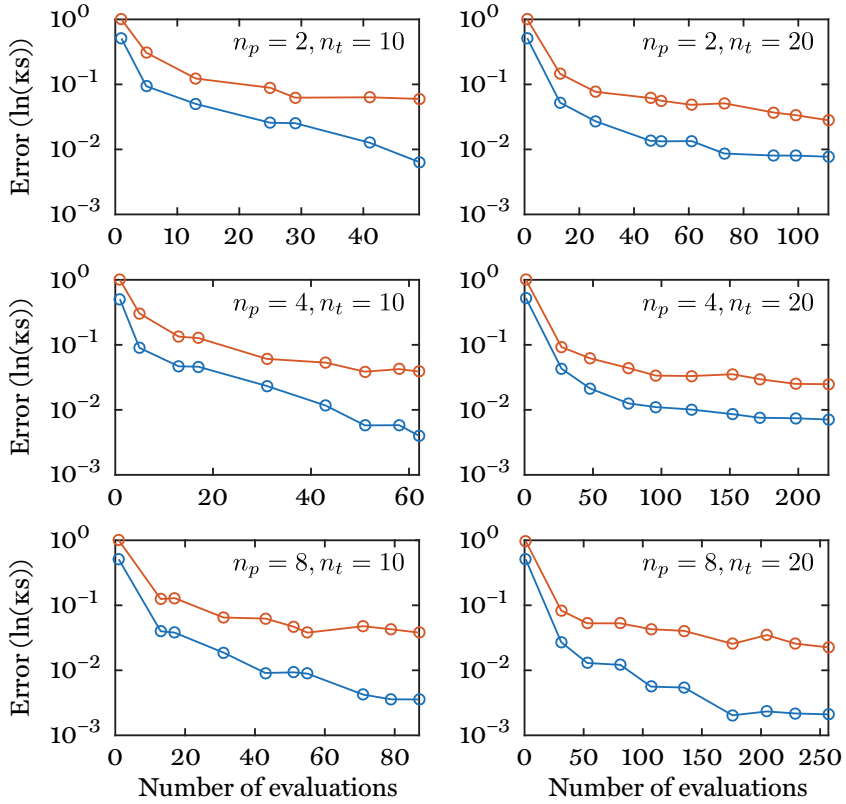

Figure 6.6: Accuracy of the proposed solution (*blue*) and direct sampling (*orange*) in the case of the end-to-end delay

shows the KS statistic on a logarithmic scale. Each plot has two lines. The blue line represents our technique. The circles on this line correspond to the steps of the interpolation process shown in Equation 6.2. They illustrate how the KS statistic computed with respect to the reference solution changes as the interpolation process adds nodes to the interpolant until a stopping condition is satisfied. Note that, in order to make the figure legible, only a subset of the actual steps is displayed. Synchronously with the blue line, that is, for the same numbers of evaluations of g, the orange line shows the error of direct sampling, which is also calculated with respect to the reference solution.

We begin by describing one particular problem chosen among those shown in the three figures. Consider, for instance, the one labeled with ★ in Figure 6.7. It can be seen that, at the very beginning, when the number of evaluations is very small, both the proposed solution and direct sampling perform poorly. The KS statistic indicates that there is a substantial mismatch between each of these two solutions and the reference one. However, as the hierarchical

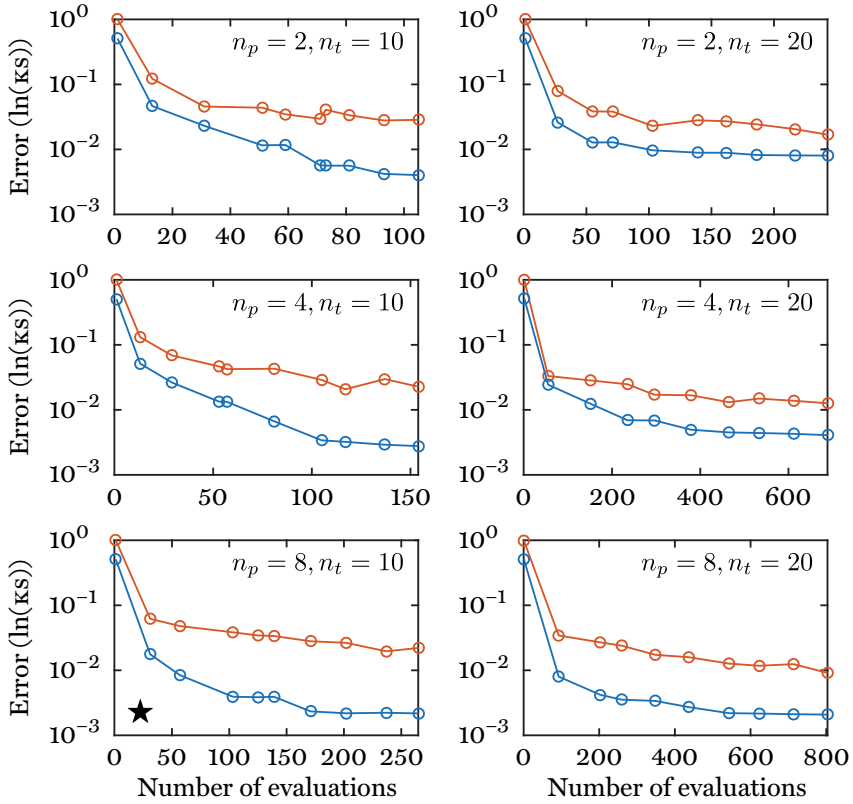

Figure 6.7: Accuracy of the proposed solution (*blue*) and direct sampling (*orange*) in the case of the total energy consumption

interpolant is being adaptively refined, our solution rapidly approaches the reference one and, by the end of the interpolation process, leaves the solution produced by direct sampling approximately an order of magnitude behind.

Studying Figure 6.6, Figure 6.7, and Figure 6.8, one can make a number of observations. Our interpolation-based approach (the blue lines) to probabilistic analysis outperforms direct sampling (the orange lines) in all the cases. This means that, given a fixed budget of computation time, the probability distributions delivered by our framework are closer to the true ones than those delivered by sampling g directly, despite the fact that the latter relies on Sobol sequences, which are a sophisticated sampling strategy. Since sampling methods try to cover the probability space impartially, the figures are a salient illustration of the difference between being adaptive and nonadaptive.

It can also be seen in the figures that, as the number of evaluations increases, the solutions computed by our technique approach the true ones. The error of the framework generally decreases more steeply than the error of di-

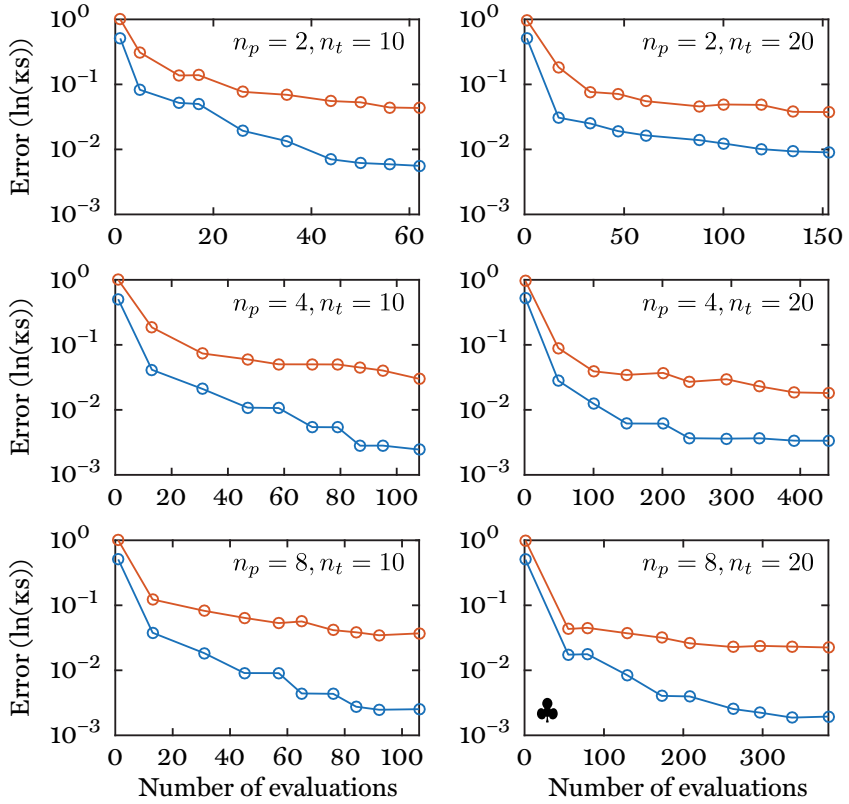

Figure 6.8: Accuracy of the proposed solution (*blue*) and direct sampling (*orange*) in the case of the maximum temperature

rect sampling. The decrease, however, tends to plateau toward the end of the interpolation process (terminated by a stopping condition). The observed behavior has two potential explanations. First, the algorithm is instructed to satisfy certain accuracy requirements (given by ϵ_a, ϵ_r, and ϵ_s), and it reasonably does not do more than what is requested. Second, since model order reduction is performed in the case of interpolation, the quantity being interpolated is not g strictly speaking; it is a low-dimensional representation of g, which already implies some information loss. Consequently, there is a certain limit on the accuracy that can be achieved, which depends on the amount of reduction.

The message of the above observations is that the designer of an electronic system can benefit substantially in terms of accuracy per computation time by switching from direct sampling to the proposed technique. If the designer's current workhorse is classical MC sampling, the switch might lead to even more dramatic savings than those shown in the figures. It is worth mentioning that

the gain is especially prominent in situations where the analysis needs to be performed many times, such as for the purpose of design-space exploration.

Remark 6.3. *The wall-clock time taken by the experiments is not reported here, as this time is irrelevant: since the evaluation of g is time consuming (see Section 6.3), the number of these evaluations is the most apposite expense indicator. For the curious reader, however, let us give an example by considering the problem labeled with ♣ in Figure 6.8. Obtaining a reference solution with 10^5 samples in parallel on 16 cores takes around two hours. Constructing an interpolant with 383 collocation nodes takes around 30 seconds (this is also the time of direct sampling with 383 samples). Sampling the interpolant 10^5 times takes less than a second. The relative computation cost of sampling an interpolant readily diminishes as the complexity of g increases; contrast it with direct sampling, whose cost grows proportionally to the evaluation time of g.*

6.10.2 Real-Life Deployment

Last but not least, we investigate the viability of deploying the proposed framework in a real environment. This means that we should couple the framework with a battle-proven simulator, used in both academia and industry, and let this simulator evaluate a real application running on a real platform.

The scenario that we consider is similar to the one depicted in Figure 6.4. The difference is that an industry-standard simulator is put in place of the "black box" on the left-hand side of the figure, and that the quantity of interest g is now the total energy. Unlike the synthetic examples discussed earlier, there is no true solution against which to compare due to the prohibitive expense of running the simulator, which is exactly why our framework is needed.

The chosen simulator is the well-known and widely used combination of Sniper [10] and McPAT [72]. The architecture that we simulate is Intel's Nehalem-based Gainestown series. Sniper is distributed with a configuration for this architecture, and we utilize this configuration without any changes. The simulated platform is set up to have three CPUs sharing one L3 cache. Regarding the application chosen for this example, it is VIPS, which is a piece of image-processing software taken from the PARSEC benchmark suite [8]. In this scenario, VIPS applies a fixed set of operations to a given image. The width and height of the image to process are considered as the uncertain parameters $\boldsymbol{u}$, and they are assumed to be uniformly distributed within certain ranges.

The deployment of the real-life example has fulfilled our expectations. The interpolation process has successfully finished and delivered an interpolant after 78 invocations of the simulator. Each such invocation takes 40 minutes on average. The probability distribution of the total energy consumption has been estimated by sampling the constructed surrogate 10^5 times. This number of samples would take around six months to obtain on our machine if we sampled

the simulator directly in parallel on 16 cores; using the technique presented in this chapter, the whole procedure has taken approximately nine hours.

6.11 Conclusion

In this chapter, we have developed a framework for system-level analysis of electronic systems whose runtime behaviors depend on uncertain parameters. The proposed approach thrives on hierarchical interpolation guided by an advanced adaptation strategy, which makes our framework general and suitable for studying various quantities that are of interest to the designer. Examples include the end-to-end delay, total energy consumption, and maximum temperature of the system. The hybrid adaptivity featured by the framework makes it particularly well suited for problems with idiosyncratic behaviors and steep response surfaces, which often arise in electronic systems due to their nature.

When provided with a means of evaluating the quantity of interest for a given outcome of the uncertain parameters and a description of the probability distribution of these parameters, the proposed framework prescribes the steps that need to be taken in order to analyze the quantity from a probabilistic perspective in a manner that is computationally efficient. Concretely, it delivers a lightweight representation that allows for a straightforward calculation of the probability distribution and other characteristics of the quantity of interest.

The performance of our technique has been evaluated by addressing a number of problems that often appear in electronic-system design. The results produced by our approach have been compared with the ones produced by an advanced sampling technique with a large number of samples. The comparison has shown that, for a fixed budget of evaluations of the quantity of interest, the framework achieves higher accuracy compared to direct sampling. Our technique has also been applied to a real-life problem, which has confirmed that the deployment of the framework in a real-life context is straightforward.

Finally, note that, even though the proposed framework has been exemplified by considering random execution times and three specific quantities of interest, it is general and can be applied in many other settings. Additionally, the approach to reliability analysis presented in Section 5.11 can benefit from the development given in this chapter by constructing surrogate representations via adaptive hierarchical interpolation instead of the PC decomposition, thereby making it possible to work with nonsmooth response surfaces.

7 Management under Workload Uncertainty

So far, we have been chiefly concerned with making uncertainty-aware decisions during various stages in the development of an electronic system, which is the main topic of the thesis. In this penultimate chapter, we go on to perform an investigation of how uncertainty can be mitigated at runtime. Specifically, we consider resource management under workload uncertainty.

7.1 Introduction

Resource management is of great importance, since it is the activity that, if done well, allows one to exploit the full potential of the computer system under consideration. However, in many cases, there is very little control over the operating environment. In particular, the actual runtime workload of a general-purpose system is rarely, if ever, known in advance. In such cases, resource management inevitably has to contend with workload uncertainty.

Workload uncertainty can be mitigated at runtime by predicting the future and acting accordingly. This is the general principle that proactive resource managers thrive on. However, accurate and useful prediction is not straightforward: modern computer systems are reasonably complex, and their resource management reasonably requires elaborate forecasting mechanisms.

Prediction traditionally falls within the scope of machine learning [46]. The field has recently received a great deal of attention due to the renaissance in neural networks [45]. This family of techniques constitutes a highly promising assistant for resource management; however, although resource management has already seen a number of applications of neural networks (to be discussed in Section 7.3), the research in this area is limited. In particular, only primitive architectures of these networks have been considered, and they have been applied to relatively simple problems. This state of affairs is unfortunate

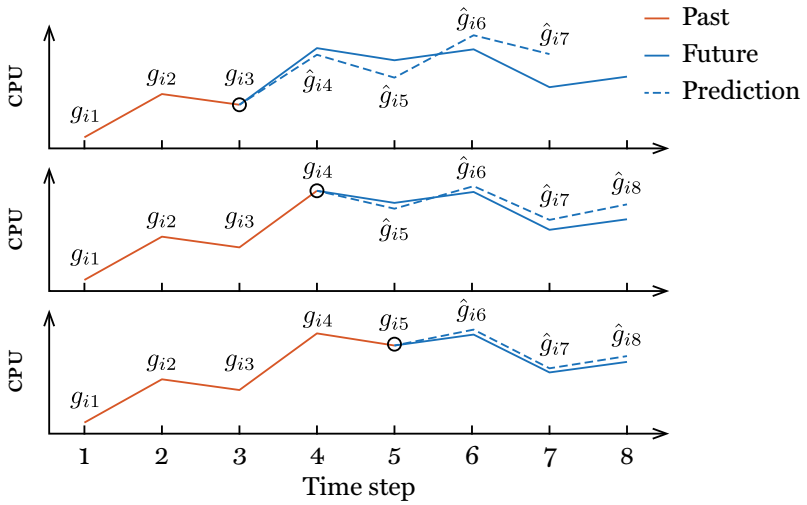

Figure 7.1: Example of predicting the CPU usage of a particular task up to four steps ahead at three different moments in time

given that neural networks have been all but revolutionary in other disciplines. Therefore, we feel strongly that more research should be conducted in order to investigate the potential aid that the recent advancements in machine learning could provide to the design of resource managers for computer systems.

In this chapter, we conduct one such body of research. More specifically, we study the usage of resources in a large system of computers and aim to predict this usage multiple steps ahead at the level of individual tasks that are executed by the machines. To this end, we use recurrent neural networks [45].

In order to give a better sense of the scenario being considered, Figure 7.1 illustrates how the aforementioned prediction is supposed to work in practice. In this example, the CPU usage of a single task running on a single machine is depicted three times; see the solid lines. The three cases correspond to three different moments in time as viewed by the resource manager of the system; see the black circles. The solid orange lines represent the history of the usage, which is known to the manager, while the solid blue lines represent the future usage, which is unknown to the manager. Our objective is then to predict this future usage for each task several steps ahead. In Figure 7.1, our potential predictions up to four steps ahead are depicted as a set of dashed blue lines.

Information about future resource usage with the level of detail (individual tasks) and foresight (multiple steps ahead) that Figure 7.1 depicts could be of great help to the resource manager of the system under consideration. In particular, the manager can more intelligently decide which machine the next incoming task should be delegated to provided that it is able to foresee how the currently active tasks will utilize the system's resources in the future.

7.2 Problem Formulation

Consider a system of computers that is serving a stream of tasks that are distributed across the individual machines of the system by a resource manager. The tasks consume certain resources, such as CPU and memory, during their execution. Let task i be characterized by a resource-usage profile defined as a sequence of n_g-dimensional measurements taken with a certain sampling interval and captured by the following $n_g \times n_{s,i}$ matrix:

$$\mathbf{G}_i = (\boldsymbol{g}_{ij})_{j=1}^{n_{s,i}} \tag{7.1}$$

where $\boldsymbol{g}_{ij} \in \mathbb{R}^{n_g}$ is the measurement taken at time step j, and $n_{s,i}$ denotes the length of the sequence. Such information is called fine grained, as it contains multiple measurements over the execution of the task as opposed to having only one aggregate measurement, such as the average or maximum value.

Suppose now that the current time step with respect to task i is j; an illustration for $j \in \{3, 4, 5\}$ is given in Figure 7.1. This means that $\boldsymbol{g}_{i1}, \ldots, \boldsymbol{g}_{ij}$ are known. Given these previous values, the objective is to estimate the next h values of the sequence, which we denote by $\hat{\boldsymbol{g}}_{i,j+1}, \ldots, \hat{\boldsymbol{g}}_{i,j+h}$; in Figure 7.1, $h = 4$. Such prediction is called long range, since it provides multiple future values as opposed to providing only one. The operation should be performed with respect to each active task of interest at each moment of interest.

In order to tackle this problem, one is supposed to learn from historical data, that is, from a data set of past profiles. Denote this data set by

$$G = \{\mathbf{G}_i : i = 1, \ldots, n_\omega\} \tag{7.2}$$

where n_ω is the total number of profiles, and $\mathbf{G}_i$ is as in Equation 7.1. Such data can be straightforwardly collected provided that the system at hand has an adequate monitoring facility deployed, which is commonplace in practice.

It should be understood that, in order for learning to be possible, the available resource-usage profiles have to have a certain structure that could be extracted and utilized for meaningful prediction. Therefore, an important question in this regard is whether real-life profiles of this kind exhibit such a structure at all. Investigating this question is part of our objective in this chapter.

7.3 Previous Work

Let us discuss a number of studies that leverage machine-learning techniques in order to facilitate resource management in computer systems.

In [23], the subject of forecasting is temperature, and the objective is attained by means of an autoregressive–moving-average model [46], resulting in an efficient thermal management strategy for multiprocessor systems. The work in [67] enhances runtime thermal management by providing an on-chip

temperature predictor based on feedforward neural networks [46]. The analysis and mitigation of the impact of process variation undertaken in [60] are facilitated by a linear regression model [46] constructed based on measurements of static power with the goal of predicting peak temperatures.

The work in [25], which is more directly relevant to the topic of this chapter, is concerned with cloud data centers. The authors propose a framework for predicting the number of virtual-machine requests together with the required amounts of CPU and memory. The framework makes use of k-means clustering [46] for identifying different types of requests, and then it utilizes Wiener filters in order to estimate the aggregate workload with respect to each identified type. Similarly to [25], the work in [53] is focused on forecasting virtual-machine requests in cloud data centers and relies on k-means clustering as the first step. Unlike [25], the main workhorse in the case of the technique in [53] is extreme learning machines, which are feedforward neural networks. An ensemble model [46] is presented in [9] targeted at predicting CPU usage in cloud environments. It relies on multiple traditional models, and the final prediction is obtained by combing these models via a scoring algorithm.

It can be seen that, in general, machine learning has been extensively utilized in order to aid resource managers of computer systems. However, as noted in Section 7.1, the most recent advancements have not yet been sufficiently explored in this context. In particular, the utility of neural networks has been studied only marginally: feedforward neural networks—which are utilized, for instance, in [53, 67]—are arguably the simplest and least powerful members of the family. However, the family is rich and potent. For instance, recurrent neural networks accompanied by adequate training techniques [45] are a salient candidate for resource management in computer systems.

In addition, note that the predictions delivered by the techniques proposed in [9, 25, 53] are coarse. They treat virtual-machine requests or computational resources as a fluid and predict the level that this fluid will attain at the next moment in time. This means that they are not capable of characterizing individual tasks executed by the machines. More generally, resource-usage prediction at the level of individual tasks, potentially multiple steps ahead, has not received adequate attention. However, prediction of this type could provide the resource manager with more detailed and foresighted information, thereby allowing for more intelligent orchestration of the system.

To summarize, only primitive architectures of neural networks have been investigated in the literature on resource management of computer systems, and only aggregate prediction has been considered so far. Therefore, there is a palpable need for further exploration and development in this direction.

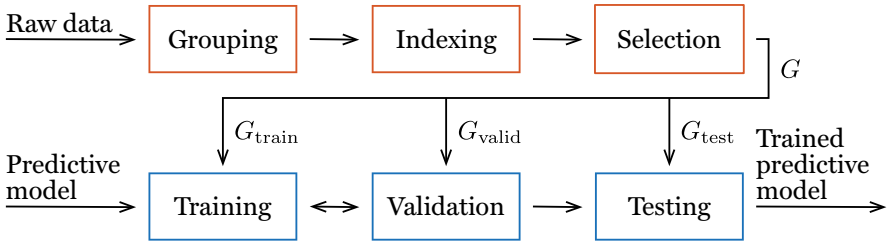

Figure 7.2: Overview of the proposed solution, including the data pipeline (*top*) and the learning pipeline (*bottom*)

7.4 Proposed Solution

Our workflow for resource-usage prediction is illustrated in Figure 7.2. First, we note that the available data generally need to be processed prior to learning, since they are likely to be given in a format that is not efficient or convenient to the subsequent calculations. With this in mind, our foremost task is to extract G defined in Equation 7.2 from the given raw data. This processing step can be seen at the top of Figure 7.2 and is explained in Section 7.4.1. Next, the resulting profiles are utilized in order to obtain an adequately trained predictive model. The modeling part is covered in Section 7.4.2 while the learning part is explained in Section 7.4.3; the latter can also be seen at the bottom of Figure 7.2. These operations are to be performed offline, whereas the trained model is supposed to be used by the resource manager at runtime in order to make predictions and subsequently take account of workload uncertainty.

7.4.1 Data Pipeline

In this subsection, we describe our pipeline for working with large data sets, which makes the data readily accessible for machine-learning applications. In order to make the exposition clearer, the pipeline is described by applying it to a real-life data set of resource-usage traces collected in a large computer cluster. To this end, we begin with a brief introduction to this data set.

The data set that we work with is the Google cluster-usage traces [94]. The traces were collected over 29 days in May 2011 and encompass more than 12 thousand machines serving requests from more than 900 users. In the data set's parlance, a user request is a job, a job comprises one or several tasks, and a task is a Linux program to be run on a single machine. Each job is assigned a unique ID, and each task of a job is given an ID that is unique within the scope of the job. Apart from other tables, the data set contains a table that records the resource usage of individual tasks with a granularity of five minutes. Each record corresponds to a specific task and a specific five-minute interval, and it provides measurements such as the average and maximum values of CPU,

memory, and disk usage. There are more than a billion records in the resource-usage table, which correspond to more than 670 thousand jobs and more than 24 million tasks with associated resource-usage profiles.

The resource-usage table is provided in the form of 500 archives. Each archive contains a single file with measurements over a certain time window. This format is inconvenient and inefficient to work with, which is addressed as described below and shown by the three topmost boxes in Figure 7.2.

In the first step (the leftmost orange box in Figure 7.2), the data from the 500 archives are distributed into separate databases so that each database contains all the data points of a particular job, resulting in as many databases as there are jobs. In order to reduce disk-space requirements, only the columns of the table that are actually used are preserved. In our experiments, these columns are the start time stamp, job ID, task ID, and average CPU usage.

In the second step (the middle orange box in Figure 7.2), an index of the profiles is created in order to be able to efficiently navigate the catalog of the databases created in the previous step. Each record in the index contains metadata about a single task, the most important of which are the task ID and the path to the corresponding database. We also include the length of the profile in the index in order to be able to efficiently filter the profiles by length.

In the last step (the rightmost orange box in Figure 7.2), a subset of the resource-usage profiles is selected using the index according to the needs of a particular learning session (to be discussed in Section 7.5) and then stored on disk. Concretely, the profiles are fetched from the databases and stored in the native format of the machine-learning library being utilized; we refer to a file in such a format as a binary file. Instead of writing all the selected profiles into one binary file, we distribute them across several files. Such a catalog of binary files is created for each of the three parts of the commonly used partition of the available data [46]: one is for training, one for validation (model selection or development), and one for testing. In the following, these three parts of G are denoted by G_{train}, G_{valid}, and G_{test}, respectively; see also Figure 7.2.

Lastly, it is common practice to standardize data before feeding them into a learning algorithm [46]. In our case, this is done along with creating the aforementioned catalogs, which requires a separate pass over the training set.

In conclusion, the benefit of the data pipeline described above is in the streamlined access to the data during one or more learning sessions. The binary files can be read efficiently as many times as needed, and they can be readily regenerated whenever the selection criteria change. It is worth noting that the artifacts of the procedures performed in the grouping and indexing steps stay the same. Additionally, the presence of multiple binary files allows for shuffling the training data at the beginning of each training epoch.

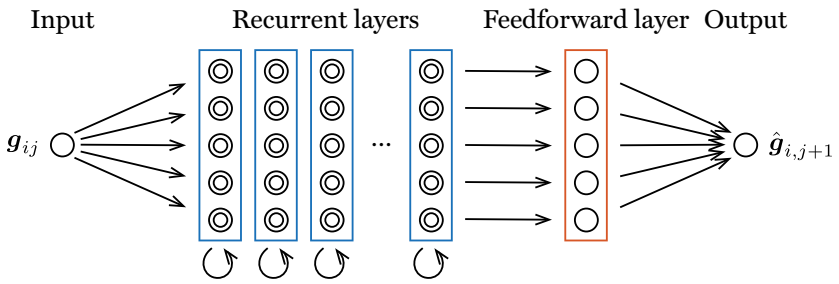

Figure 7.3: Schematic representation of the predictive model

7.4.2 Predictive Model

As emphasized in Section 7.1 and Section 7.2, our objective is to assess the applicability of the latest advancements in neural networks [45] to fine-grained long-range prediction of resource usage in computer systems. The architectures of neural networks are very diverse. However, since the data that we study are inherently sequential, it is natural to found our predictive model on the basis of recurrent neural networks [45], which are designed for such cases.

A schematic representation of our model can be seen in Figure 7.3. Note that many of the actual connections between the parts of the model are simplified or not shown at all in order to make the figure legible. In the rest of this subsection, we describe each of the parts. A number of important operational aspects of the model will be covered in the next subsection, Section 7.4.3.

The input to the predictive model is a single n_g-dimensional data point, which can be seen on the left-hand side of Figure 7.3. Similarly, the output is a single n_g-dimensional data point, which is depicted on the right-hand side of Figure 7.3. The input $\boldsymbol{g}_{ij}$ is the value of the resource usage of a single task at time step j, and the output $\hat{\boldsymbol{g}}_{i,j+1}$ is a one-step-ahead prediction of this usage.

The core of the model is formed by a number of recurrent layers. In Figure 7.3, these layers are represented by a group of blue boxes. The network can be made as many layers deep as needed. Each layer is composed of a number of units, which are depicted as double circles in Figure 7.3. The number of layers and the number of units per layer are denoted by n_l and n_u, respectively.

Units are the smallest processing elements. The key characteristics of a unit of a recurrent layer are that the unit has internal memory, and that it has access to its previous output. There are different types of recurrent units, defining how these units let data flow through them and update their memory. One notable unit is the long short-term memory (LSTM) unit [49], which has been designed to overcome the problems of traditional recurrent neural networks, such as the vanishing gradient during training. The recurrent layers of our predictive model are LSTM layers, that is, layers composed of LSTM units.

Each recurrent layer is enhanced by a dropout mechanism [122], which is active only during training. This mechanism gives control over the regularization of the model and is meant to prevent potential overfitting [46]. For future reference, denote the probability of dropping an output of a layer by ρ_{drop}.

The output of the last recurrent layer is typically a large tensor, which is proportional in size to the number of units in the layer. Each entry of such a tensor can be regarded as a feature that the network has extracted and activated in accordance with the resource-usage profile that is currently being fed to the model. The task now is to combine these features in order to produce a single prediction. To this end, we mount a feedforward layer with a linear activation function on top of the last recurrent layer, which is depicted as an orange box in Figure 7.3. Unlike the recurrent layers, which feature highly nonlinear transformations, this layer performs an affine transformation.

To summarize, we have described a predictive model that is composed of a number of recurrent layers and a feedforward layer. Due to its internal memory, the model is capable of efficiently taking account of the entire past of the resource-usage profile in question when predicting the future of this profile. Let us now discuss how the predictive model is meant to be used.

7.4.3 Learning Pipeline

Recall that the output of the data pipeline described in Section 7.4.1 is the resource-usage data set G, which is split into three parts: G_{train}, G_{valid}, and G_{test}. The three parts are utilized during the following three steps of what we refer to as the learning pipeline: training, validation, and testing. This learning pipeline is discussed below and illustrated at the bottom of Figure 7.2.

We begin with training. The model depicted in Figure 7.3 has a large number of parameters that have to be learned; they are primarily various weights and biases inside the layers. For this purpose, G_{train} is utilized. The training is undertaken via backpropagation through time using stochastic gradient descent [45] with the objective of minimizing a certain loss function, which we shall specify shortly. There are two aspects that need to be discussed first.

The first concerns the way a single profile is fed to the predictive model. To begin with, the internal memory of the model is nullified. Next, recall that each profile contains several data points (see Equation 7.1), and note that two profiles generally have different lengths ($n_{s,i} \neq n_{s,j}$), since the execution times of two tasks are likely to differ. With this in mind, each profile is fed to the model using a technique called dynamic unrolling. An illustration is depicted in Figure 7.4, in which the schematic representation given in Figure 7.3 has been simplified even further and rotated 90° counterclockwise. It can be seen that the model has been replicated as many times as there are data points in the profile. However, it is still the same model, and all the replicas share the same parameters and the same internal memory. It can also be observed in

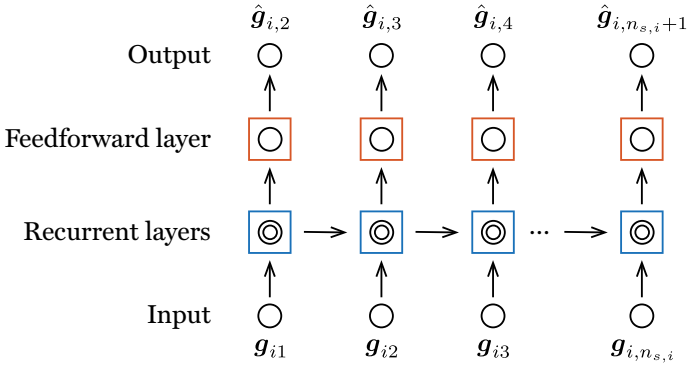

Figure 7.4: Feeding a resource-usage profile into the predictive model

Figure 7.4 how information flows from one time step to the next one, which implicitly gives the model access to the whole history at each time step.

Now, it is not efficient to feed only one resource-usage profile at a time due to the inevitable overhead imposed by the computations involved. These computations should be performed in batches; that is, n_b profiles should be fed simultaneously where n_b is referred to as the batch size. Since, in general, $n_{s,i} \neq n_{s,j}$, it is not possible to stack multiple arbitrary profiles into a single tensor. In order to circumvent this problem, we resort to bucketing. Specifically, each profile is put into one of many buckets that is chosen based on the profile's length. When a bucket collecting profiles of length from some $n_{s,l}$ to $n_{s,r}$ has received n_b profiles, it pads the ones that are shorter than $n_{s,r}$ with zeros and emits a tensor of size $n_b \times n_{s,r} \times n_g$ to be consumed by the model.

The loss function that is minimized during training is the mean squared error (MSE) of one-step predictions over the whole batch. The correct prediction for the very last time step, which goes beyond the time window of the profiles in question, is assumed to be zero. For instance, in Figure 7.4, there is no $\boldsymbol{g}_{i,n_{s,i}+1}$ against which to compare $\hat{\boldsymbol{g}}_{i,n_{s,i}+1}$; thus, $\boldsymbol{g}_{i,n_{s,i}+1}$ is assumed to be zero.

Let us now discuss validation, which corresponds to the middle blue box in Figure 7.2. Similarly to other nontrivial models, the one presented in Section 7.4.2 has a number of hyperparameters. Examples include the number of recurrent layers n_l, number of units per layer n_u, and probability of dropout ρ_{drop}. Unlike ordinary parameters, which are optimized during training as described earlier, hyperparameters are set prior to training and kept unchanged thereafter. From the examples given, it is apparent that hyperparameters can have a profound impact. Hence, they should be tuned with great care.

The validation set G_{valid} is used to assess the performance of the model after it has been trained (using G_{train} as usual) with different configurations of the model's hyperparameters. Similarly to training, the error metric is the

MSE, and it is beneficial to perform validation in batches. The trained model that has the best performance on G_{valid} is then chosen as the one to use.

Despite all the techniques employed to speed up training, it is still a time-consuming operation. This means that a brute-force search in the space of the hyperparameters for the best configuration is impractical; therefore, a more intelligent strategy should be followed. In our workflow, we use the Hyperband algorithm [71]. Instead of adaptively choosing new configurations to evaluate, which is the case with many algorithms of this kind, Hyperband adaptively allocates resources to configurations chosen at random, which is demonstrated to be an efficient strategy. In particular, the algorithm allows for a reduction in computation time by promptly eliminating overtly inadequate configurations of the hyperparameters. In this context, *resources* refers to a user-defined measure of how extensively a configuration is to be exercised. For instance, it can be the amount of wall-clock time spent or the number of training steps taken; in our experiments documented in Section 7.5, we use the latter.

Let us now turn to testing; see the rightmost blue box in Figure 7.2. After a trained model has been selected during the validation step, it has to be reassessed [46]: one cannot aver that the error with respect to G_{valid} is a good estimate of the generalization error due to selection bias—we have deliberately chosen the configuration that has the best performance on G_{valid}.

In order to attain an unbiased evaluation, the testing set G_{test} is utilized. Similarly to training and validation, the MSE is considered as a measure of quality, and bucketing is utilized. However, unlike with training and validation, the error is calculated in a more elaborate way as follows. Recall first that our objective is making long-range predictions of resource usage; see Section 7.2. Note also that, in training and validation, we are concerned with what happens only one step ahead. The reason for this is that we would like to have the highest throughput possible during training and validation, since they are performed many times. Testing, on the other hand, is done only once, and it is during testing that we make and assess multiple-step-ahead predictions.

In order to calculate long-range predictions, we use refeeding: at time step j, the predicted value $\hat{\boldsymbol{g}}_{i,j+1}$ of the resource usage of task i is fed to the model as if it was the actual resource usage $\boldsymbol{g}_{i,j+1}$ at time step $j+1$, which is not yet known at time step j. At this point, it might be helpful to consider the example in Figure 7.1. The process continues until all the desired h future values have been estimated. It is natural to expect that a more accurate one-step-ahead prediction will lead to a more accurate multiple-step-ahead one.

Consider now how a trained predictive model might be used in practice. Potentially, at each time step j, the next h values of the resource usage of task i, that is, $\hat{\boldsymbol{g}}_{i,j+1}, \ldots, \hat{\boldsymbol{g}}_{i,j+h}$, might have to be estimated. Therefore, in order to test the model properly, we have to consider all the time steps of the profile in question and make h predictions at each time step. One important aspect to note is that the state of the model's memory should be saved before computing $\hat{\boldsymbol{g}}_{i,j+1}, \ldots, \hat{\boldsymbol{g}}_{i,j+h}$ at time step j and then restored before feeding $\boldsymbol{g}_{i,j+1}$ in

order to advance to time step $j + 1$. This is because the memory becomes contaminated when one feeds predictions instead of observations into the model.

At this point, the main aspects of our workflow, which is illustrated in Figure 7.2, have been discussed. The output of the workflow is a predictive model that has been trained on G_{train}, validated on G_{valid}, and tested on G_{test}.

7.5 Experimental Results

In this section, we assess our workflow for making fine-grained long-range predictions of resource usage in computer systems, which is described in Section 7.4. All the experiments are conducted on a GNU/Linux machine equipped with 8 Intel Core i7-3770 3.4 GHz processors, 24 GB of RAM, and an HDD of 500 GB. All the source code, configuration files, and input data used in the experiments are available online at [21]; see also [22].

7.5.1 Data Pipeline

Recall that we study the Google cluster-usage traces [94], which are introduced in Section 7.4.1. Without loss of generality, we focus on the consumption of one particular resource, namely CPU; thus, $n_g = 1$ in Equation 7.1. In this regard, the data set provides two apposite pieces of information: the average and maximum CPU usage over five-minute intervals; we extract the former.

The grouping and indexing steps, which are described in Section 7.4.1 and depicted in Figure 7.2, take around 57 hours (no parallelism). Since they have to be performed only once, their computational cost can be considered negligible. Regarding the selection step, we filter those resource-usage profiles that contain 5–50 data points; hence, $n_{s,i} \in [5, 50]$ in Equation 7.1. Such profiles constitute about 74% of the total number of profiles (about 18 out of 24 million). We experiment with a random subset of two million profiles, which is roughly 11% of the 5–50 resource-usage profiles; thus, $n_\omega = 2 \times 10^6$ in Equation 7.2. The data sets G_{train}, G_{valid}, and G_{test} constitute 70%, 10%, and 20% of G, respectively. The process of fetching this number of profiles and storing them on disk takes around four hours. Recall that this operation has to be repeated only when the selection criteria change, which happens rarely.

7.5.2 Learning Pipeline

The training step (see the leftmost blue box in Figure 7.2) is configured as follows. Ten buckets are used according to the following partition:

$$5 < 6 < 7 < 8 < 9 < 10 < 15 < 20 < 30 < 40 \leq 50.$$

For instance, the first and last buckets collect profiles with $n_{s,i} = 5$ and $n_{s,i} \in [40, 50]$, respectively. The batch size n_b is set to 64. The optimization algorithm

employed for minimizing the loss function is Adam [63], which is an adaptive technique. The algorithm is applied with its default settings.

Let us now discuss the validation step, which corresponds to the middle blue box in Figure 7.2. The hyperparameters being considered are the number of recurrent layers n_l (the blue boxes in Figure 7.3), number of units per layer n_u (the double circles in Figure 7.3), and probability of dropout ρ_{drop}; these hyperparameters are introduced in Section 7.4.2. Specifically, we let n_l be in $\{1, 2, 3, 4, 5\}$, n_u in $\{100, 200, 400, 800, 1600\}$, and ρ_{drop} in $\{0, 0.25, 0.5\}$, which yields 75 different combinations in total. The candidate solutions are explored by means of the Hyperband algorithm with its default settings; this algorithm is introduced in Section 7.4.3. The maximum budget granted to one configuration is set to four training epochs, which correspond to $4 \times 0.7 \times 2 \times 10^6 = 5.6 \times 10^6$ resource-usage profiles or $5.6 \times 10^6 \div 64 = 87{,}500$ training steps.

The aforementioned exploration, which encompasses both training and validation, takes roughly 15 days to finish. During this process, we run up to four learning sessions in parallel, which typically keeps all eight cores busy. It should be noted that, since the training, validation, and testing data sets have been cached on disk as a result of the data pipeline described in Section 7.4.1, individual learning sessions do not have any overhead in this regard. It is also worth noting that the machine utilized in these experiments has no modern GPUs; therefore, there is a great deal of room for performance improvement.

The results of the validation step are reported in Table 7.1. The table shows the MSEs of the top 10 and bottom 10 configurations of the hyperparameters as measured using G_{valid} ($0.1 \times 2 \times 10^6 = 2 \times 10^5$ profiles). The total number of distinct cases sampled by Hyperband is 62. It can be seen that the error changes rapidly at the bottom of the table and slows down toward the top. Relative to the least accurate trained model (rank 62), the error drops by around 22% in the bottom 10 and by around 3% in the top 10 configurations. The overall improvement in accuracy with respect to the bottommost configuration is about 36%, which indicates that the validation step is highly beneficial.

The best trained predictive model (rank 1) is found to have the following hyperparameters: $n_l = 3$, $n_u = 1600$, and $\rho_{\text{drop}} = 0$. In general, larger architectures tend to outperform smaller ones, which is expected. However, there are complex configurations at the bottom of Table 7.1 as well, which could be due to the dropout mechanism engaged in those cases. To elaborate, in these experiments, the impact of dropouts is found to be mostly neutral or negative; compare, for instance, the candidates of ranks 2 and 60. This could be due to the amount of training data being sufficient for regularizing the model.

Table 7.1 also shows an estimate of the memory required by each configuration, which includes the trainable parameters and internal state of the model. It can be seen that, if memory usage is a concern, one could trade a small decrease in accuracy for a considerable saving in memory. For example, the fourth best solution requires around 85% less memory than the first one.

Table 7.1: Accuracy and memory requirements of the top 10 and bottom 10 configurations of the hyperparameters of the predictive model

Rank	n_c	n_u	ρ_{drop}	Error (MSE)	Memory (MB)
1	3	1600	0.00	0.3148	197.81
2	4	1600	0.00	0.3154	276.76
3	3	1600	0.25	0.3159	197.81
4	2	800	0.25	0.3194	30.14
5	5	200	0.00	0.3205	6.01
6	2	1600	0.50	0.3207	118.87
7	5	800	0.00	0.3251	89.97
8	3	800	0.00	0.3257	50.08
9	1	1600	0.25	0.3278	39.92
10	2	800	0.00	0.3316	30.14
53	1	200	0.00	0.3834	0.72
54	1	100	0.25	0.3851	0.21
55	3	100	0.50	0.3871	0.92
56	2	100	0.25	0.3888	0.56
57	5	400	0.25	0.3891	23.01
58	5	800	0.50	0.3918	89.97
59	4	100	0.50	0.3978	1.28
60	4	1600	0.50	0.4115	276.76
61	5	400	0.50	0.4385	23.01
62	5	100	0.25	0.4894	1.63

After the validation step, the best trained predictive model is taken to the testing step; see Figure 7.2. The solution is assessed extensively by predicting resource usage for individual tasks multiple steps ahead at each time step of the profiles in G_{test} ($0.2 \times 2 \times 10^6 = 4 \times 10^5$ profiles). In these experiments, we predict four time steps into the future; therefore, $h = 4$ in Section 7.2. This elaborate and mostly sequential procedure takes approximately 18 hours.

In order to attain a better assessment of the accuracy of the chosen trained model, we employ an alternative solution referred to as the reference solution. This alternative solution is based on random walk. It postulates that the best prediction of what will happen tomorrow is what happens today plus an optional random offset, which we set to zero. In other words, the next value for the resource usage of each analyzed task is estimated to be the current one, which subsequently results in four identical predictions at each time step.

The results of the testing step can be seen in Figure 7.5, which shows the MSEs of our solution (the blue line) and the reference one (the orange line) with respect to G_{test}. The magnitude of the errors of our predictive model suggests that the amount of regularity present in the data is not sufficient for making highly accurate resource-usage predictions. Nevertheless, one can observe in

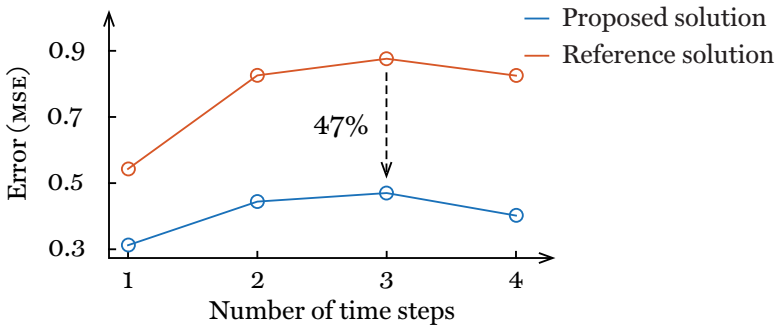

Figure 7.5: Accuracy of the proposed and reference solutions for predicting resource usage for individual tasks multiple steps ahead

Figure 7.5 that, relative to the reference solution, the trained model provides an error reduction of approximately 47% at each of the four time steps. This observation indicates that some structure does exist, and that this structure can be identified and utilized in order to make educated predictions.

7.6 Conclusion

In this chapter, we have elaborated on the possibility of mitigating workload uncertainty at runtime and, more specifically, presented our experience of working with state-of-the-art recurrent neural networks in the context of fine-grained long-range prediction of resource usage in computer systems. Our workflow—which starts by making the data readily available for learning and finishes by making predictions with respect to individual tasks multiple time steps into the future—has been described in detail and applied to a large data set of real resource-usage profiles collected in a computer cluster.

The obtained results suggest that the fine-grained resource-usage profiles that have been studied possess a certain structure, and that this structure can be extracted by advanced machine-learning techniques and subsequently utilized for making educated predictions. This detailed information could be of great use to the resource manager of the computer system under consideration, allowing the manager to more intelligently orchestrate the system.

8 Conclusion

Modern electronic systems are complex, operate in complex environments, and perform complex tasks. Even in the deterministic case, analysis and design of such systems are highly challenging endeavors. However, their difficulty is escalated even further by uncertainty that accompanies electronic systems.

The objective of this thesis has been to assist the designer of electronic systems by providing tools that would allow for effective and efficient quantification and mitigation of uncertainty that stems from the fabrication process, workload, and aging. In this final chapter, we recapitulate the main outcomes of the work that has been presented and outline directions for future work.

8.1 Present Work

We have developed a number of uncertainty-aware system-level techniques for analyzing and designing electronic systems. The features that are common to our solutions are efficiency, generality, and straightforwardness of application.

In Chapter 3, we have elaborated on the deterministic scenario, which has served as an adequate starting point for the developments in the subsequent chapters. We have presented an auxiliary transformation of the temperature model that simplifies certain operations associated with temperature calculations. Furthermore, we have proposed a novel approach to dynamic steady-state power and temperature analysis. The high accuracy and speed of our technique make it readily applicable inside intensive design-space-exploration loops, which has been illustrated by performing a reliability optimization.

In Chapter 4, we have considered the variability that occurs in process parameters as a result of process variation across silicon wafers. In this context, we have presented a versatile statistical framework for inferring the aforementioned variability by means of indirect measurements, which can potentially

be incomplete and corrupted by noise. The ability to work with such measurements implies low costs, since it might obviate the need for deployment of any specialized test structures, and in scenarios where such structures are already available, it might require actuating only a small subset of them.

In Chapter 5, we have presented a methodology for studying diverse system-level quantities that are of interest to the designer, but that are also uncertain due to process variation. Examples of such quantities include transient and dynamic steady-state power and temperature profiles of the system at hand as well as the system's maximum temperature and total energy consumption. Our approach treats the quantity being analyzed as an opaque object, which makes it flexible and convenient to use, and delivers a surrogate for this object that allows the designer to efficiently estimate various probabilistic characteristics of this quantity. The efficiency makes the proposed technique suitable for exploring the design space. This has been exemplified in the context of energy optimization with reliability-related constraints, in which reliability analysis has been enhanced to consider process uncertainty.

In Chapter 6, we have described another technique for probabilistic analysis of system-level quantities that are of interest to the designer. In this case, we have striven to provide an adequate characterization of the variability that originates from workload. This variability tends to be less regular than the one that stems from the fabrication process and, therefore, necessitates a different treatment. Our approach allows the designer to analyze important quantities that are subject to workload variation, such as end-to-end delays, energy consumption, and peak temperatures. The experimental results have demonstrated that, given a fixed budget of evaluations of the quantity under consideration, the designer can benefit substantially in terms of accuracy per computation time by utilizing the proposed technique instead of direct sampling.

In Chapter 7, we have elaborated on mitigation of workload uncertainty at runtime in the context of resource management. Specifically, we have performed an early investigation into the applicability of advanced prediction techniques from machine learning to the problem of fine-grained long-range forecasting of resource usage in large computer systems. The results of this investigation indicate that the real-life data that have been studied possess certain regularities, and that these regularities can be modeled by advanced techniques and subsequently utilized for making sensible predictions.

8.2 Future Work

As with many other bodies of research, the one presented in this thesis has a beginning and an end. The latter rarely implies that the subject under consideration has been exhaustively studied and requires no further attention. Instead, it merely means that the research project has come to a certain conclusion and made a certain contribution to the understanding or treatment of this subject.

Naturally, this can leave a number of research questions open and is likely to pose new questions, which is also the case with our work.

Before we discuss particulars, let us first make one general remark about our research. It would be valuable to investigate how our uncertainty-aware techniques perform in practice. Although we have conducted such an investigation in the case of the interpolation-based technique presented in Chapter 6, this study is relatively small and has been done in an academic setting. Industrial settings are different, and they might help to reveal our blind spots.

In the experiments reported in Chapter 4, Chapter 5, and Chapter 6, we assume certain probability distributions. Even though our assumptions are reasonable, it would be helpful to assess our solutions using probability distributions that are estimated based on real-life measurements. In particular, as noted in the chapters, the assumed correlations affect not only the accuracy but also the feasibility of applying our techniques. The concern about feasibility is particularly acute in situations where the curse of dimensionality is an issue, and this curse could be an issue in Chapter 5 and Chapter 6. Recall that the former leverages the polynomial chaos (PC) decomposition while the latter makes use of adaptive hierarchical interpolation. Having identified a number of problematic real-life use cases, one could investigate how the proposed techniques should be configured or even extended in order to make them applicable in those cases. For instance, the attentive reader might have noted that the control over anisotropy that is provided by the framework based on PC expansions has not been sufficiently explored in the experimental results. However, if used properly, it could effectively mitigate the curse of dimensionality.

It would also be meaningful to perform an elaborate comparison and explore the boundaries between the techniques proposed in Chapter 5 and Chapter 6. Our default policy is to use the former for process uncertainty and the latter for workload uncertainty, which is motivated by the response surfaces that are characteristic of the two types of uncertainty. However, this does not mean that one of the techniques cannot be successfully used in the primary application area of the other. For instance, workload uncertainty might lead to smooth variations, in which case a polynomial surrogate might be beneficial.

The early investigation reported in Chapter 7 is arguably the most prominent direction for further development within the scope of this thesis. Mitigation of uncertainty at runtime is considered advantageous, since one has access to large amounts of relevant and previously inaccessible-by-definition information and, therefore, can make more educated decisions. As emphasized in Chapter 7, modern architectures of neural networks accompanied by modern approaches to training have great potential, and resource management is an important beneficiary in this regard. This potential, however, has yet to materialize. Extensive research in this direction would be highly desirable.

A Appendix

In the appendix, we define a number of concepts and describe a number of techniques that are utilized extensively throughout the thesis.

A.1 Linear Algebra

Any symmetric matrix $\mathbf{X} \in \mathbb{R}^{n \times n}$ admits the eigendecomposition [87]. In this particular case, the decomposition takes the following form:

$$\mathbf{X} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}} \tag{A.1}$$

where $\mathbf{U} \in \mathbb{R}^{n \times n}$ is an orthogonal matrix of the eigenvectors of $\mathbf{X}$, and

$$\mathbf{\Lambda} = \operatorname{diag}(\lambda_1, \ldots, \lambda_n) = \begin{pmatrix} \lambda_1 & \cdots & 0 \\ \cdots & \cdots & \cdots \\ 0 & \cdots & \lambda_n \end{pmatrix} \in \mathbb{R}^{n \times n}$$

is a diagonal matrix of the eigenvalues of $\mathbf{X}$. If, in addition, $\mathbf{X}$ is a positive semidefinite matrix, the eigenvalues $\{\lambda_i\}_{i=1}^{n}$ are non-negative.

A.2 Probability Theory

Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a (complete) probability space where Ω is a set of outcomes, $\mathcal{F} \subseteq 2^{\Omega}$ is a σ-algebra on Ω, and $\mathbb{P} : \mathcal{F} \to [0, 1]$ is a probability measure [35]. The σ-algebra represents a set of events, and the probability measure assigns probabilities to these events. A real-valued random variable defined on $(\Omega, \mathcal{F}, \mathbb{P})$ is an $\mathcal{F}$-measurable function $x : \Omega \to \mathbb{R}$. A random variable x is uniquely characterized by its distribution function F, which is also known as the cumulative distribution function (CDF), defined by

$$F(y) = \mathbb{P}(x \leq y) = \mathbb{P}(\{\omega : \omega \in \Omega, x(\omega) \leq y\}).$$

The expectation of x is given by

$$\mathbb{E}x = \int_{\mathbb{R}} y \, \mathrm{d}F(y) = \int_{\Omega} x(\omega) \mathrm{d}\mathbb{P}(\omega) .$$

If $\mathbb{E}x = 0$, x is called centered. The variance of x is given by

$$\mathbb{V}\mathrm{ar}(x) = \mathbb{E}(x - \mathbb{E}x)^2 = \mathbb{E}x^2 - (\mathbb{E}x)^2. \tag{A.2}$$

If $\mathbb{V}\mathrm{ar}(x) = 1$, x is called normalized. If x is both centered and normalized (that is, $\mathbb{E}x = 0$ and $\mathbb{V}\mathrm{ar}(x) = 1$), x is called standardized.

The above quantities are well defined only when the corresponding integrals are finite. An example of a vector space of random variables defined on $(\Omega, \mathcal{F}, \mathbb{P})$ whose expectations and variances are finite is

$$L^2(\Omega, \mathcal{F}, \mathbb{P}) = \left\{ x : \int_{\Omega} |x(\omega)|^2 \mathrm{d}\mathbb{P}(\omega) < \infty \right\}, \tag{A.3}$$

the Hilbert space of square-integrable random variables [56]. The inner product in this vector space is defined by

$$\langle x_1, x_2 \rangle = \mathbb{E}(x_1 x_2),$$

and the norm is defined by

$$\|x\|_2 = \sqrt{\langle x, x \rangle}$$

for any x, x_1, and x_2 in $L^2(\Omega, \mathcal{F}, \mathbb{P})$.

If the distribution function F of a random variable x is continuous, x is said to be a continuous random variable. If, moreover, F is absolutely continuous, x is said to have a density function f, which is also known as the probability density function (PDF). In such a case,

$$F(y) = \int_{-\infty}^{y} f(z) \mathrm{d}z.$$

Other types of random variables, such as discrete random variables, are of little relevance to this thesis; therefore, they are not covered here.

Suppose now that there is a set of n random variables $\{x_i\}_{i=1}^n$. For convenience, the variables are arranged in a vector $\boldsymbol{x} = (x_i)_{i=1}^n : \Omega \to \mathbb{R}^n$, which is called a random vector and which can also be viewed as a single random variable taking values in $\mathbb{R}^n$. The variables obey a common distribution known as the joint distribution, and the distribution of any subset of the variables is called a marginal distribution. The individual variables are referred to as mutually independent if their joint distribution function F factorizes as follows:

$$F(\boldsymbol{x}) = \prod_{i=1}^{n} F_i(x_i)$$

where F_i is the marginal distribution of x_i for $i = 1, \ldots, n$.

Analogously to the one-dimensional case, $\boldsymbol{x}$ has an expectation and a variance (if the integrals are finite) as well as a PDF (if the CDF is absolutely continuous). In addition, the covariance of x_i and x_j is defined as follows:

$$\mathbb{C}\mathrm{ov}\left(x_i, x_j\right) = \mathbb{E}((x_i - \mathbb{E}x_i)(x_j - \mathbb{E}x_j)).$$

If $\mathbb{C}\mathrm{ov}(x_i, x_j) = 0$, x_i and x_j are referred to as uncorrelated. Furthermore, the covariance matrix of $\boldsymbol{x}$ is given by

$$\mathbb{C}\mathrm{ov}\left(\boldsymbol{x}\right) = \left(\mathbb{C}\mathrm{ov}\left(x_i, x_j\right)\right)_{i=1,j=1}^{n,n}, \tag{A.4}$$

which is a positive semidefinite matrix by definition. Lastly, a special case of the covariance matrix is the correlation matrix

$$\mathbb{C}\mathrm{orr}\left(\boldsymbol{x}\right) = \mathrm{diag}\left(\mathbb{C}\mathrm{ov}\left(\boldsymbol{x}\right)\right)^{-\frac{1}{2}} \mathbb{C}\mathrm{ov}\left(\boldsymbol{x}\right) \mathrm{diag}\left(\mathbb{C}\mathrm{ov}\left(\boldsymbol{x}\right)\right)^{-\frac{1}{2}} \tag{A.5}$$

where $\mathrm{diag}(\cdot)$ extracts the diagonal elements of its argument and yields a diagonal matrix. It can be shown that the correlation matrix is the covariance matrix of a normalized version of $\boldsymbol{x}$, that is, of

$$\left(\frac{x_i}{\sqrt{\mathbb{V}\mathrm{ar}(x_i)}}\right)_{i=1}^{n}.$$

The joint distribution of $\boldsymbol{x} : \Omega \to \mathbb{R}^n$ can be unambiguously specified by a set of n one-dimensional marginal distributions and an n-dimensional copula [79]. The copula is a uniform distribution function on $[0, 1]^n$ that captures the dependencies between the n individual variables contained in $\boldsymbol{x} = (x_i)$.

A.3 Bayesian Statistics

Let x be an uncertain parameter that we would like to characterize. To this end, the following information is at our disposal: (a) a set of observations Y of a quantity y that depends on x; (b) a data model f that describes the relationship between x and y, which is denoted by $y = f(x)$; and (c) prior knowledge (or beliefs) about x. A natural solution is Bayes' theorem [42]

$$p(x|Y) \propto p(Y|x)p(x)$$

where p denotes a PDF. In this context, $p(Y|x)$ is known as the likelihood function; it accommodates the data model f and yields the relative likelihood of observing the data set Y given the parameter x, that is, given a particular assignment of the uncertain parameter x. The distribution that corresponds to $p(x)$ is known as the prior (distribution) of x; it represents the knowledge about x that is available prior to any observations. The distribution that corresponds

to $p(x|Y)$ is known as the posterior (distribution) of x; it yields the relative likelihood of the parameter x given the observations gathered in Y and taking the prior knowledge about x into consideration. The posterior is an exhaustive solution to our problem: having constructed $p(x|Y)$, the required statistics about x can be trivially estimated by taking samples from this posterior.

In practice, the posterior is unlikely to belong to any of the common families of probability distributions, which is due in part to the data model involved in the likelihood function. Therefore, the sampling procedure is not straightforward. In order to circumvent this difficulty, one usually relies on Markov chain Monte Carlo sampling [42]. In this case, an ergodic Markov chain with the stationary distribution equal to the target posterior distribution is constructed and then utilized for exploring the probability space.

A popular technique in this regard is the Metropolis–Hastings algorithm [42] where the chain is constructed via sampling from a computationally convenient distribution known as the proposal (distribution). Each sample drawn from the proposal is passed through the posterior in order to calculate its posterior probability, which is then used to decide whether the sample should be accepted or rejected. A rejection means that the sequence of samples advances using the last accepted sample—as if it was drawn once again. The acceptance strategy of the algorithm pushes the produced chain of samples toward regions of high posterior probability, which results in an adequate approximation of the posterior after a sufficient number of steps, depending on the starting point of the chain and the efficiency of the moves that have been made.

A.4 Probability Transformation

There are three probability transformations that are utilized in the thesis.

The first transformation $\mathbb{T}_1$ is the usual integral transformation. The technique allows one to transition from a random variable x_1 with a distribution function F_1 to a random variable x_2 with a distribution function F_2 as

$$x_2 = \mathbb{T}_1(x_1) = (F_2^{-1} \circ F_1)(x_1) = F_2^{-1}(F_1(x_1)$$

where F_2^{-1} is the inverse of F_2, and the equality should be understood in distribution. In order for F_2^{-1} to exist, F_2 is assumed to be a strictly increasing function. By applying F_1 to x_1, we obtain a random variable with a uniform distribution on $[0, 1]$, and, by applying F_2^{-1} to such a uniform random variable, we obtain a random variable distributed according to the target F_2.

The second transformation $\mathbb{T}_2$ uses the Karhunen–Loève (KL) decomposition [43, 120] and, more specifically, its discrete version, which is also known as principal component analysis [46]. The technique is based on the eigendecomposition given in Equation A.1 applied to Equation A.5, and it transforms potentially correlated random variables into linearly uncorrelated ones. Conversely, the approach allows one to transition from a random vector $\boldsymbol{x}_1$ with

n_1 linearly uncorrelated components to an n_1-dimensional random vector $\boldsymbol{x}_2$ with a prescribed correlation structure. Having obtained the decomposition in Equation A.1, the relationship between the two vectors is as follows:

$$\boldsymbol{x}_2 = \mathbb{T}_2(\boldsymbol{x}_1) = \mathbf{U}\boldsymbol{\Lambda}^{\frac{1}{2}}\boldsymbol{x}_1 \tag{A.6}$$

where the eigenvalues contained in $\boldsymbol{\Lambda}$ correspond to the variances induced to the uncorrelated variables when the transition is made. When $\boldsymbol{x}_2$ obeys a multivariate Gaussian distribution, the individual variables in $\boldsymbol{x}_1$ follow the standard Gaussian distribution and are mutually independent, and vice versa.

In addition, the eigendecomposition in Equation A.1 provides a means of model order reduction. The intuition is that, since the variables in $\boldsymbol{x}_2$ are correlated, $\boldsymbol{x}_2$ can be recovered sufficiently well from a small subset of the variables in $\boldsymbol{x}_1$. Specifically, we are to select the smallest subset of $\boldsymbol{x}_1$ such that its cumulative contribution to the variance of $\boldsymbol{x}_2$ is above a certain threshold. Formally, assuming that $\{\lambda_i\}_{i=1}^{n_1}$ in $\boldsymbol{\Lambda}$ are sorted in the descending order and given a threshold $\eta \in (0, 1]$, which is the fraction of the variance to be preserved, we are to identify the smallest $n_2 \leq n_1$ such that

$$\frac{\sum_{i=1}^{n_2} \lambda_i}{\sum_{i=1}^{n_1} \lambda_i} \geq \eta.$$

Then, given a vector $\boldsymbol{x}_1$ in the reduced space $\mathbb{R}^{n_1}$, the corresponding vector $\boldsymbol{x}_2$ in the original space $\mathbb{R}^{n_2}$ can be lossily reconstructed as follows:

$$\boldsymbol{x}_2 \approx \mathbb{T}_2(\boldsymbol{x}_1) = \mathbf{U}\tilde{\boldsymbol{\Lambda}}^{\frac{1}{2}}\boldsymbol{x}_1. \tag{A.7}$$

In this formula, $\tilde{\boldsymbol{\Lambda}} \in \mathbb{R}^{n_2 \times n_1}$ is a truncated version of $\boldsymbol{\Lambda} \in \mathbb{R}^{n_2 \times n_2}$ in Equation A.1: the matrix contains only the first n_1 columns of $\boldsymbol{\Lambda}$. When η is sufficiently large, the dropped variables are considered insignificant or redundant.

The third transformation $\mathbb{T}_3$ builds upon the previous two transformations and allows one to tackle the following problem. Let $\boldsymbol{x}_1 : \Omega \to \mathbb{R}^{n_1}$ be a random vector whose joint distribution is known and is the product of the marginal distributions $\{F_{1i}\}_{i=1}^{n_1}$ of the individual variables; therefore, the variables are independent. Let also $\boldsymbol{x}_2 : \Omega \to \mathbb{R}^{n_2}$ be a random vector whose joint distribution is unknown, and where the only available information about $\boldsymbol{x}_2$ is the marginal distributions $\{F_{2i}\}_{i=1}^{n_2}$ and correlation matrix $\mathbb{C}\mathrm{orr}(\boldsymbol{x}_2)$, which are not sufficient to recover the joint distribution in general. The goal is to transform $\boldsymbol{x}_1$ into a random vector that has the same marginal distributions and the same correlation matrix as $\boldsymbol{x}_2$ and thereby closely approximates $\boldsymbol{x}_2$.

Making use of the first two transformations described above, we obtain a potential solution, which is loosely denoted as follows:

$$\boldsymbol{x}_2 \approx \mathbb{T}_3(\boldsymbol{x}_1) = (\mathbb{T}_1 \circ \mathbb{T}_2 \circ \mathbb{T}_1)(\boldsymbol{x}_1). \tag{A.8}$$

First, using $\mathbb{T}_1$, the n_1 independent variables $\boldsymbol{x}_1$ are transformed into n_1 independent uniform variables and then into n_1 independent standard Gaussian

variables, which should be understood elementwise. Second, using $\mathbb{T}_2$, the n_1 independent standard Gaussian variables are transformed into n_2 dependent Gaussian variables with a correlation matrix that is carefully constructed based on the knowledge about $\boldsymbol{x}_2$. Third, using $\mathbb{T}_1$, the n_2 dependent Gaussian variables are transformed into n_2 dependent uniform variables and then into n_2 dependent variables with the marginal distributions $\{F_{2i}\}_{i=1}^{n_2}$ and correlation matrix $\mathbb{C}\mathrm{orr}(\boldsymbol{x}_2)$, which approximate the n_2 dependent variables $\boldsymbol{x}_2$.

The auxiliary correlation matrix mentioned above is constructed using the Nataf model described in [74]; see also [70]. In fact, the two outermost transformations in Equation A.8 without model order reduction are often referred to as the Nataf transformation. The technique operates under the assumption that the copula of $\boldsymbol{x}_2$ is elliptical. In the general case, it is an approximation.

A.5 Numerical Integration

In numerical integration, the integral of a function $f : \mathbb{R} \to \mathbb{R}$

$$I = \int_{\mathbb{R}} f(x)\mathrm{d}x$$

is approximated as

$$I \approx \mathcal{Q}_i^1(f) = \sum_{j \in \mathcal{J}_i^1} f(x_{ij}) w_{ij},$$

which is a summation of the function's values computed at prescribed points and multiplied by prescribed weights. This type of paring of a set of points and a set of weights is called a quadrature. In the notation $\mathcal{Q}_i^1$, the superscript 1 indicates that it is a one-dimensional quadrature, and the subscript $i \in \mathbb{N}_0$ indicates the level of the quadrature. In the above formula,

$$\mathcal{X}_i^1 = \left\{x_{ij} : j \in \mathcal{J}_i^1\right\} \subset \mathbb{R} \text{ and}$$
$$\mathcal{W}_i^1 = \left\{w_{ij} : j \in \mathcal{J}_i^1\right\} \subset \mathbb{R}$$

are the points and weights of the quadrature, respectively, and $\mathcal{J}_i^1 \subset \mathbb{N}_0$ is an index set. The cardinality of $\mathcal{J}_i^1$ depends on i and is denoted by $n_i = \#\mathcal{J}_i^1$.

The level i is the index of the quadrature in the corresponding family of quadratures with increasing precision. *Precision* refers to the maximum order of polynomials that the quadrature integrates exactly [47]. To give an example, consider Gaussian quadratures, which is a broad class of quadratures incorporating many families. The precision of a Gaussian quadrature with n_i points is $2n_i - 1$ [47]. In other words, a Gaussian quadrature with n_i points is exact for polynomials of order up to $2n_i - 1$. It is a remarkable property of Gaussian quadratures, which makes them especially efficient and hence popular.

Different families of quadratures can have different relations between n_i and i. Even within the scope of the same family, the integration grid can be

made to grow differently with respect to i in order to attain certain properties, such as being nested. For our purposes, it is sufficient to mention one type of growth: slow linear growth. In this case, $n_i = i + 1$. Assuming slow linear growth, the previous paragraph can be extended by stating that a Gaussian quadrature with n_i points is exact for polynomials of order up to $2i + 1$.

In multiple dimensions, the integral of a function $f : \mathbb{R}^n \to \mathbb{R}$

$$I = \int_{\mathbb{R}^n} f(\boldsymbol{x}) \mathrm{d}\boldsymbol{x}$$

is approximated as

$$I \approx \mathcal{Q}_{\boldsymbol{i}}^n (f) = \sum_{\boldsymbol{j} \in \mathcal{J}_{\boldsymbol{i}}^n} f(\boldsymbol{x}_{\boldsymbol{ij}}) w_{\boldsymbol{ij}}$$

where $\boldsymbol{i} = (i_k) \in \mathbb{N}_0^n$ and $\boldsymbol{j} = (j_k) \in \mathbb{N}_0^n$. In the above formula,

$$\begin{aligned} \mathcal{X}_{\boldsymbol{i}}^n &= \{\boldsymbol{x}_{\boldsymbol{ij}} : \boldsymbol{j} \in \mathcal{J}_{\boldsymbol{i}}^n\} \subset \mathbb{R}^n \text{ and} \\ \mathcal{W}_{\boldsymbol{i}}^n &= \{w_{\boldsymbol{ij}} : \boldsymbol{j} \in \mathcal{J}_{\boldsymbol{i}}^n\} \subset \mathbb{R} \end{aligned}$$

and the points and weights, respectively, and $\mathcal{J}_{\boldsymbol{i}}^n \subset \mathbb{N}_0^n$ is an index set. The cardinality of $\mathcal{J}_{\boldsymbol{i}}^n$ dependents on both n and $\boldsymbol{i}$ and is denoted by $n_{\boldsymbol{i}} = \#\mathcal{J}_{\boldsymbol{i}}^n$.

The foundation of an n-dimensional quadrature $\mathcal{Q}_{\boldsymbol{i}}^n$ is a set of one-dimensional counterparts of various levels. The most straightforward construction of such a quadrature is the full tensor product

$$\mathcal{Q}_{\boldsymbol{i}}^n = \bigotimes_{k=1}^n \mathcal{Q}_{i_k}^1, \tag{A.9}$$

in which case

$$\begin{aligned} \boldsymbol{x}_{\boldsymbol{ij}} &= (x_{i_k j_k})_{k=1}^n, \\ w_{\boldsymbol{ij}} &= \prod_{k=1}^n w_{i_k j_k}, \\ \mathcal{J}_{\boldsymbol{i}}^n &= \mathcal{J}_{i_1}^1 \times \cdots \times \mathcal{J}_{i_n}^1, \text{ and} \\ n_{\boldsymbol{i}} &= \#\mathcal{J}_{\boldsymbol{i}}^n = \prod_{k=1}^n \#\mathcal{J}_{i_k}^1 = \prod_{k=1}^n n_{i_k}. \end{aligned} \tag{A.10}$$

It can be seen that, in this case, the growth of the number of points with respect to the number of dimensions is exponential. In low dimensions, the growth is manageable; however, in high dimensions, the situation changes dramatically, as the number of points produced by this approach can easily explode.

To give an example [47], suppose that each one-dimensional quadrature has only four points ($n_i = 4$). Then, in 10 stochastic dimensions ($n = 10$), the number of multivariate points becomes 1,048,576 ($n_{\boldsymbol{i}} = n_i^n = 4^{10}$), which

is far beyond being affordable. Moreover, it can be shown that most of the points obtained by the full tensor product do not contribute to the asymptotic accuracy and, therefore, are a waste of computation time. In particular, if the integrand under consideration is a polynomial whose total order is constrained according to a certain strategy, the full tensor product cannot take this information into account. Consequently, a different construction technique should be utilized in the case of high-dimensional integration problems.

An alternative construction is the Smolyak algorithm [101]; see also [36, 47, 64, 68]. The algorithm is a central technique in the field of not only integration but also interpolation; the latter is elaborated on in Appendix A.6. In the context of integration, the algorithm combines a range of one-dimensional quadratures in such a way that the resulting grid is tailored to be exact only for a certain polynomial subspace. Such grids are called sparse grids, and they allow for a significant reduction in the number of points and thus the subsequent work. For instance, in the example given earlier, the number of points would be only 1581, which is a drastic decrease in computation time.

To begin with, define

$$\begin{aligned}
&\mathcal{Q}^1_{-1} = 0, \\
&\Delta\mathcal{Q}^1_i = \mathcal{Q}^1_i - \mathcal{Q}^1_{i-1}, \text{ and} \\
&\Delta\mathcal{Q}^n_{\boldsymbol{i}} = \bigotimes_{k=1}^{n} \Delta\mathcal{Q}^1_{i_k}.
\end{aligned}$$

Smolyak's formula of level l_q is then as follows:

$$\mathcal{Q}^n_{l_q} = \bigoplus_{\boldsymbol{i}\in\mathcal{I}^n_{l_q}} \Delta\mathcal{Q}^n_{\boldsymbol{i}}. \tag{A.11}$$

In the original (isotropic) formulation of the Smolyak algorithm,

$$\mathcal{I}^n_{l_q} = \{\boldsymbol{i} : \boldsymbol{i} \in \mathbb{N}^n_0, \|\boldsymbol{i}\|_1 \leq l_q\} \tag{A.12}$$

where $\|\cdot\|_1$ stands for the Manhattan norm. The index set $\mathcal{I}^n_{l_q}$ is called the total-order index set [4, 36], and its cardinality can be calculated as follows:

$$\#\mathcal{I}^n_{l_q} = \binom{n+l_q}{n} = \frac{(n+l_q)!}{n!\,l_q!}. \tag{A.13}$$

Note that Equation A.11 reduces to Equation A.9 if we let

$$\mathcal{I}^n_{l_q} = \{\boldsymbol{i} : \boldsymbol{i} \in \mathbb{N}^n_0, \|\boldsymbol{i}\|_\infty \leq l_q\}.$$

Using Equation A.11, the integral in question is approximated as

$$I \approx \mathcal{Q}^n_{l_q}(f) = \sum_{j\in\mathcal{J}^n_{l_q}} f(\boldsymbol{x}_j) w_j.$$

Note that, for convenience, the points and weights of $\mathcal{Q}_{l_q}^n$ are indexed using a single one-dimensional set denoted by $\mathcal{J}_{l_q}^n \subset \mathbb{N}_0$. Note also that, even though the level l_q is not indicated in the notation of points and weights, it should be understood that points and weights are generally different for different levels.

Lastly, consider the following more general integral:

$$I = \int_{\mathbb{R}^n} f(\boldsymbol{x}) \mathrm{d}F(\boldsymbol{x}).$$

In this case, f is integrated with respect to a measure $F : \mathbb{R}^n \to \mathbb{R}$ [35] that does not necessarily correspond to the usual Lebesgue measure, which is used in the earlier examples. Since integrating with respect to a certain measure is a very frequent operation, there are families of quadratures that are designed to automatically take this aspect into account in the most common scenarios. For instance, the Gauss–Hermite family is suitable for integrating with respect to the standard Gaussian measure, which can be seen in Equation A.24.

It is also worth emphasizing that quadratures are generally precomputed and tabulated, since they do not depend on the function being integrated.

A.6 Hierarchical Interpolation

Let $f : [0,1] \to \mathbb{R}$ be a function in $\mathcal{C}([0,1])$, which is the space of continuous functions defined on $[0,1]$. The function is approximated by virtue of the following interpolation formula:

$$f \approx \mathcal{A}_i^1(f) = \sum_{j \in \mathcal{J}_i^1} f(x_{ij}) e_{ij}. \tag{A.14}$$

In the notation $\mathcal{A}_i^1$, the superscript 1 indicates the dimensionality, and the subscript $i \in \mathbb{N}_0$ indicates the level of the interpolant. In the above formula,

$$\mathcal{X}_i = \left\{x_{ij} : j \in \mathcal{J}_i^1\right\} \subset [0,1] \text{ and}$$
$$\mathcal{E}_i = \left\{e_{ij} : j \in \mathcal{J}_i^1\right\} \subset \mathcal{C}\left([0,1]\right)$$

are collocation nodes and basis functions, respectively, and $\mathcal{J}_i^1$ is an index set. The cardinality of $\mathcal{J}_i^1$ depends on i and is denoted by $n_i = \#\mathcal{J}_i^1$. In this context, the subscript $j \in \mathcal{J}_i^1$ is referred to as the order of a node or a function.

Let us now turn to the multidimensional case. Let $f : [0,1]^n \to \mathbb{R}$ be a function in $\mathcal{C}([0,1]^n)$, which is the space of continuous functions defined on $[0,1]^n$. The function is approximated as

$$f \approx \mathcal{A}_{\boldsymbol{i}}^n(f) = \sum_{\boldsymbol{j} \in \mathcal{J}_{\boldsymbol{i}}^n} f(\boldsymbol{x_{ij}}) e_{\boldsymbol{ij}}$$

where $\boldsymbol{i} = (i_k) \in \mathbb{N}_0^n$ and $\boldsymbol{j} = (j_k) \in \mathbb{N}_0^n$. In the above formula,

$$\mathcal{X}_{\boldsymbol{i}} = \{\boldsymbol{x_{ij}} : \boldsymbol{j} \in \mathcal{J}_{\boldsymbol{i}}^n\} \subset [0,1]^n \text{ and}$$
$$\mathcal{E}_{\boldsymbol{i}} = \{e_{\boldsymbol{ij}} : \boldsymbol{j} \in \mathcal{J}_{\boldsymbol{i}}^n\} \subset \mathcal{C}\left([0,1]^n\right)$$

are the nodes and functions, respectively, and $\mathcal{J}_{\boldsymbol{i}}^n \subset \mathbb{N}_0^n$ is an index set. The cardinality of $\mathcal{J}_{\boldsymbol{i}}^n$ depends on both n and $\boldsymbol{i}$ and is denoted by $n_{\boldsymbol{i}} = \#\mathcal{J}_{\boldsymbol{i}}^n$.

Similarly to Appendix A.5, the construction of $\mathcal{A}_{\boldsymbol{i}}^n$ can be based on the full tensor product of n one-dimensional interpolants as follows:

$$\mathcal{A}_{\boldsymbol{i}}^n = \bigotimes_{k=1}^{n} \mathcal{A}_{i_k}^1, \tag{A.15}$$

in which case

$$\begin{aligned} \boldsymbol{x}_{\boldsymbol{ij}} &= (x_{i_k j_k})_{k=1}^n, \\ e_{\boldsymbol{ij}} &= \bigotimes_{k=1}^{n} e_{i_k j_k}, \text{ and} \end{aligned}$$

the index set $\mathcal{J}_{\boldsymbol{i}}^n$ is the same as the one in Equation A.10. The main observation with respect to the above construction is similar to the one made in Appendix A.5: the number of collocation nodes grows exponentially as the number of dimensions increases. Nevertheless, Equation A.15 serves well as a building block for a more efficient algorithm, which we discuss next.

This algorithm is the Smolyak algorithm introduced in Appendix A.5. In this context, the algorithm combines a number of one-dimensional interpolants in such a way that the resulting interpolant preserves the approximating power of the full tensor product only for a certain polynomial subspace, which allows it to drastically reduce the number of collocation nodes.

Define

$$\begin{aligned} &\mathcal{A}_{-1}^1 = 0, \\ &\Delta\mathcal{A}_i^1 = \mathcal{A}_i^1 - \mathcal{A}_{i-1}^1, \text{ and} \\ &\Delta\mathcal{A}_{\boldsymbol{i}}^n = \bigotimes_{k=1}^{n} \Delta\mathcal{A}_{i_k}^1. \end{aligned} \tag{A.16}$$

The Smolyak algorithm of level l_s is then

$$\mathcal{A}_{l_s}^n = \sum_{\boldsymbol{i} \in \mathcal{I}_{l_s}^n} \Delta\mathcal{A}_{\boldsymbol{i}}^n = \mathcal{A}_{l_s-1}^n + \sum_{\boldsymbol{i} \in \Delta\mathcal{I}_{l_s}^n} \Delta\mathcal{A}_{\boldsymbol{i}}^n \tag{A.17}$$

where

$$\begin{aligned} \mathcal{I}_{l_s}^n &= \{\boldsymbol{i} : \boldsymbol{i} \in \mathbb{N}_0^n, \|\boldsymbol{i}\|_1 \leq l_s\} \text{ and} \\ \Delta\mathcal{I}_{l_s}^n &= \{\boldsymbol{i} : \boldsymbol{i} \in \mathbb{N}_0^n, \|\boldsymbol{i}\|_1 = l_s\}. \end{aligned} \tag{A.18}$$

It can be seen in Equation A.17 that a Smolyak interpolant can be efficiently refined: the work done in order to attain one level can be entirely recycled in

order to go to the next one. Regarding the collocation nodes, define

$$\begin{aligned}
&\mathcal{X}^1_{-1} = \emptyset, \\
&\Delta\mathcal{X}^1_i = \mathcal{X}^1_i \setminus \mathcal{X}^1_{i-1}, \text{ and} \\
&\Delta\mathcal{X}^n_{\boldsymbol{i}} = \Delta\mathcal{X}^1_{i_1} \times \cdots \times \Delta\mathcal{X}^1_{i_n}.
\end{aligned}$$

The collocation nodes are then as follows:

$$\mathcal{X}^n_{l_s} = \bigcup_{\boldsymbol{i} \in \mathcal{I}^n_{l_s}} \Delta\mathcal{X}^n_{\boldsymbol{i}} = \mathcal{X}^n_{l_s - 1} \cup \bigcup_{\boldsymbol{i} \in \Delta\mathcal{I}^n_{l_s}} \Delta\mathcal{X}^n_{\boldsymbol{i}}. \tag{A.19}$$

The above formula also shows how the collocation nodes of one level are related to the collocation nodes of the previous level. The sparsity and incremental refinement of the Smolyak approach are remarkable properties *per se*; however, the approach can be taken even further, as we discuss next.

It can be seen in Equation A.19 that it is beneficial to the refinement to have $\mathcal{X}^1_{i-1}$ be entirely included in $\mathcal{X}^1_i$, since, in that case, the cardinality of

$$\mathcal{X}^n_{l_s} \setminus \mathcal{X}^n_{l_s - 1} = \bigcup_{\boldsymbol{i} \in \Delta\mathcal{I}^n_{l_s}} \Delta\mathcal{X}^n_{\boldsymbol{i}}$$

derived from Equation A.19 decreases. To express this idea in words, the values of f obtained at lower levels can be reused in order to attain higher levels if the grid grows without discarding its previous structure. With this in mind, the rule used for generating successive sets of points $\{\mathcal{X}^1_i\}$ should be chosen to be nested, that is, in such a way that $\mathcal{X}^1_i$ contains all the nodes present in $\mathcal{X}^1_{i-1}$.

The last step is to rewrite Equation A.17 in a hierarchical form. To this end, we require interpolants of higher levels to exactly represent interpolants of lower levels. In one dimension, this means that

$$\mathcal{A}^1_{i-1}(f) = \mathcal{A}^1_i(\mathcal{A}^1_{i-1}(f)). \tag{A.20}$$

The above condition can be satisfied by an appropriate choice of collocation nodes and basis functions. Assuming that Equation A.20 holds and using Equation A.14 and Equation A.16,

$$\Delta\mathcal{A}^1_i(f) = \sum_{j \in \Delta\mathcal{J}^1_i} \left(f(x_{ij}) - \mathcal{A}^1_{i-1}(f)(x_{ij})\right) e_{ij}$$

where

$$\Delta\mathcal{J}^1_i = \left\{j : j \in \mathcal{J}^1_i, x_{ij} \in \Delta\mathcal{X}^1_i\right\}.$$

The above summation is over $\Delta\mathcal{X}^1_i$ due to the fact that the difference in the parentheses is zero when $x_{ij} \in \mathcal{X}^1_{i-1}$, since $\mathcal{X}^1_{i-1} \subset \mathcal{X}^1_i$. In multiple dimensions,

$$\Delta\mathcal{A}^n_{\boldsymbol{i}}(f) = \sum_{\boldsymbol{j} \in \Delta\mathcal{J}^n_{\boldsymbol{i}}} \left(f(\boldsymbol{x_{ij}}) - \mathcal{A}^n_{l_s - 1}(f)(\boldsymbol{x_{ij}})\right) e_{\boldsymbol{ij}} \tag{A.21}$$

where $l_s = \|\boldsymbol{i}\|_1$ and

$$\Delta\mathcal{J}_{\boldsymbol{i}}^n = \{\boldsymbol{j} : \boldsymbol{j} \in \mathcal{J}_{\boldsymbol{i}}^n, \boldsymbol{x}_{\boldsymbol{ij}} \in \Delta\mathcal{X}_{\boldsymbol{i}}^n\}. \tag{A.22}$$

The delta

$$\Delta f(\boldsymbol{x}_{\boldsymbol{ij}}) = f(\boldsymbol{x}_{\boldsymbol{ij}}) - \mathcal{A}_{l_s-1}^n(f)(\boldsymbol{x}_{\boldsymbol{ij}}) \tag{A.23}$$

is referred to as a hierarchical surplus. When increasing the interpolation level, this surplus is the difference between the actual value of the target function at a new collocation node and the approximation of this value computed by the hierarchical interpolant that has been constructed so far.

The final formula for (nonadaptive) hierarchical interpolation is obtained by substituting Equation A.21 into Equation A.17. The result is

$$\begin{aligned} f \approx \mathcal{A}_{l_s}^n(f) &= \sum_{\boldsymbol{i}\in\mathcal{I}_{l_s}^n} \sum_{\boldsymbol{j}\in\Delta\mathcal{J}_{\boldsymbol{i}}^n} \Delta f(\boldsymbol{x}_{\boldsymbol{ij}}) e_{\boldsymbol{ij}} \\ &= \mathcal{A}_{l_s-1}^n(f) + \sum_{\boldsymbol{i}\in\Delta\mathcal{I}_{l_s}^n} \sum_{\boldsymbol{j}\in\Delta\mathcal{J}_{\boldsymbol{i}}^n} \Delta f(\boldsymbol{x}_{\boldsymbol{ij}}) e_{\boldsymbol{ij}} \end{aligned}$$

where $\Delta f(\boldsymbol{x}_{\boldsymbol{ij}})$ is computed according to Equation A.23.

A.7 Polynomial Chaos

Let $L^2(\Omega, \mathcal{F}, \mathbb{P})$ be as in Equation A.3. Let also $G \subset L^2(\Omega, \mathcal{F}, \mathbb{P})$ be the Gaussian Hilbert space [56] spanned by n mutually independent standard Gaussian random variables, which are denoted by $\boldsymbol{x} : \Omega \to \mathbb{R}^n$. Since the variables are independent and standard, they form an orthonormal basis in G, and G is n-dimensional. The variables induce a probability measure on $\mathbb{R}^n$, and the corresponding distribution function F is standard Gaussian given by

$$\mathrm{d}F(\boldsymbol{y}) = (2\pi)^{-\frac{n}{2}} \exp\left(-\frac{{\|\boldsymbol{y}\|_2}^2}{2}\right) \mathrm{d}\boldsymbol{y}. \tag{A.24}$$

The inner product in the vector space of functions over G is defined as

$$\langle g, h\rangle = \int_{\mathbb{R}^n} g(\boldsymbol{y}) h(\boldsymbol{y}) \mathrm{d}F(\boldsymbol{y}) \tag{A.25}$$

for any g and h in this space, and the norm is defined as

$$\|g\|_2 = \sqrt{\langle g, g\rangle}.$$

Let $\Psi_{l_c}(G)$ be the space of n-variate polynomials over G such that the total order of each polynomial is at most $l_c \in \mathbb{N}_0$. The space $\Psi_{l_c}(G)$ can be constructed as a span of n-variate Hermite polynomials [36, 68]

$$\Psi_{l_c}(G) = \operatorname{span}\left(\left\{\psi_{\boldsymbol{i}}(\boldsymbol{y}) : \boldsymbol{i} \in \mathcal{I}_{l_c}^n, \boldsymbol{y} \in G\right\}\right)$$

where $\boldsymbol{i} = (i_k) \in \mathbb{N}_0^n$, the index set $\mathcal{I}_{l_c}^n$ is the one in Equation A.12, and

$$\psi_{\boldsymbol{i}}(\boldsymbol{y}) = \prod_{k=1}^{n} \psi_{i_k}(y_k).$$

In the above formulae, $\psi_{i_k} : \mathbb{R} \to \mathbb{R}$ is a normalized one-dimensional Hermite polynomial of order i_k. An important property of the polynomials $\{\psi_{\boldsymbol{i}}\}$ is that they are mutually orthonormal with respect to F, which means that

$$\langle \psi_{\boldsymbol{i}}, \psi_{\boldsymbol{j}} \rangle = \delta_{\boldsymbol{ij}} \tag{A.26}$$

for any $\boldsymbol{i} = (i_k) \in \mathbb{N}_0^n$ and $\boldsymbol{j} = (j_k) \in \mathbb{N}_0^n$ where

$$\delta_{\boldsymbol{ij}} = \prod_{k=1}^{n} \delta_{i_k j_k},$$

and $\delta_{i_k j_k}$ is the Kronecker delta. In addition, the polynomials are centered with respect to F; therefore, the inner product in Equation A.25 applied to two polynomials yields their correlation, which is zero due to the orthogonality. Furthermore, the inner product applied to a polynomial with itself yields the variance of that polynomial, which is unity due to the normality.

Define

$$\begin{aligned} &\Delta\Psi_{-1} = \emptyset \text{ and} \\ &\Delta\Psi_i = \Psi_i(G) \cap \Psi_{i-1}(G)^{\perp}. \end{aligned}$$

The vector spaces $\{\Delta\Psi_i\}_{i=0}^{\infty}$ are mutually orthogonal, closed subspaces of $L^2(\Omega, \mathcal{F}, \mathbb{P})$. Since our scope of interest is restricted to functions of $\boldsymbol{x}$, $\mathcal{F}$ is assumed to be generated by $\boldsymbol{x}$. Then, by the Cameron–Martin theorem,

$$L^2(\Omega, \mathcal{F}, \mathbb{P}) = \bigoplus_{i=0}^{\infty} \Delta\Psi_i.$$

The decomposition is called the Wiener chaos decomposition; however, it is more often referred to as the classical polynomial chaos (PC) decomposition.

The PC decomposition implies that any $f \in L^2(\Omega, \mathcal{F}, \mathbb{P})$ admits the following infinite expansion with respect to the polynomial basis:

$$f = \sum_{\boldsymbol{i} \in \mathbb{N}_0^n} \hat{f}_{\boldsymbol{i}} \psi_{\boldsymbol{i}}$$

where the equality should be understood in mean square. In practice, this infinite expansion should be truncated, which we denote by

$$f \approx \mathcal{C}_{l_c}^n(f) = \sum_{\boldsymbol{i} \in \mathcal{I}_{l_c}^n} \hat{f}_{\boldsymbol{i}} \psi_{\boldsymbol{i}}$$

where $\mathcal{I}_{l_c}^n$ is a certain index set, frequently the one given in Equation A.12. In the notation $\mathcal{C}_{l_c}^n$, the superscript n indicates that there are n random variables, and the subscript l_c indicates the level of the truncated expansion. Each coefficient $\hat{f}_{\boldsymbol{i}}$ is found by taking the inner product with respect to the corresponding polynomial $\psi_{\boldsymbol{i}}$ on both sides of the above equation and making use of the orthogonality property shown in Equation A.26. The result is

$$\hat{f}_{\boldsymbol{i}} = \langle f, \psi_{\boldsymbol{i}} \rangle \,. \tag{A.27}$$

This operation is referred to as a spectral projection.

The classical PC decomposition can be generalized to other families of probability distributions besides the Gaussian one. Many distributions directly correspond to certain families of orthogonal polynomials, which can be found in the Askey scheme of orthogonal polynomials. A distribution that does not have such a correspondence can be transformed into one of those that do using techniques such as the one shown in Appendix A.4. Another solution is to construct a custom polynomial basis using the Gram–Schmidt process. Note that the machinery of PC expansions is applicable to discrete probability distributions as well. The interested reader is referred to [120] for further discussions.

Bibliography

[1] T. Adam, K. Chandy, and J. Dickson. "A comparison of list schedules for parallel processing systems." In: *Communications of the ACM* 17.12 (Dec. 1974), pp. 685–690.

[2] S. Asmussen and P. Glynn. *Stochastic simulation: Algorithms and analysis*. Springer-Verlag New York, 2007.

[3] M. Bao, A. Andrei, P. Eles, and Z. Peng. "Temperature-aware idle time distribution for energy optimization with dynamic voltage scaling." In: *Design, Automation, and Test in Europe Conference*. Mar. 2010, pp. 21–26.

[4] J. Beck, F. Nobile, L. Tamellini, and R. Tempone. "Implementation of optimal Galerkin and collocation approximations of PDEs with random coefficients." In: *ESAIM: Proceedings* 33 (Oct. 2011), pp. 10–21.

[5] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, and M. Olivieri. "MPARM: Exploring the multi-processor SoC design space with SystemC." In: *Journal of VLSI Signal Processing Systems* 41.2 (Sept. 2005), pp. 169–182.

[6] S. Bhardwaj, S. Vrudhula, P. Ghanta, and Y. Cao. "Modeling of intradie process variations for accurate analysis and optimization of nanoscale circuits." In: *Design Automation Conference*. July 2006, pp. 791–796.

[7] S. Bhardwaj, S. Vrudhula, and A. Goel. "A unified approach for full chip statistical timing and leakage analysis of nanoscale circuits considering intradie process variations." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.10 (Oct. 2008), pp. 1812–1825.

[8] C. Bienia. "Benchmarking modern multiprocessors." PhD thesis. Princeton University, Jan. 2011.

[9] J. Cao, J. Fu, M. Li, and J. Chen. "CPU load prediction for cloud environment based on a dynamic ensemble model." In: *Software: Practice and Experience* 44.7 (July 2014), pp. 793–804.

[10] T. Carlson, W. Heirman, and L. Eeckhout. "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations." In: *International Conference for High Performance Computing, Networking, Storage and Analysis*. Nov. 2011, 52:1–52:12.

[11] S. Chandra, K. Lahiri, A. Raghunathan, and S. Dey. "Variation-aware system-level power analysis." In: *IEEE Transactions on Very Large Scale Integration Systems* 18.8 (Aug. 2010), pp. 1173–1184.

[12] A. Chandrakasan, W. Bowhill, and F. Fox. *Design of high-performance microprocessor circuits*. Wiley-IEEE Press, 2000.

[13] S.-C. Chang, S.-Y. Deng, and J. Lee. "Electrical characteristics and reliability properties of MOSFET with Dy2O3 gate dielectric." In: *Applied Physics Letters* 89.5 (Aug. 2006).

[14] M. Chaudhry, T. Ling, A. Manzoor, S. Hussain, and J. Kim. "Thermal-aware scheduling in green data centers." In: *ACM Computing Surveys* 47.3 (Feb. 2015), 39:1–39:48.

[15] L. Cheng, P. Gupta, C. Spanos, K. Qian, and L. He. "Physically justifiable die-level modeling of spatial variation in view of systematic across wafer variability." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30.3 (Mar. 2011), pp. 388–401.

[16] *Collection of supplementary materials*. Embedded Systems Laboratory. Nov. 2017. URL: https://www.ida.liu.se/labs/eslab/alumni/ivanukhov/research/2012-SDTA/.

[17] *Collection of supplementary materials*. Embedded Systems Laboratory. Nov. 2017. URL: https://www.ida.liu.se/labs/eslab/alumni/ivanukhov/research/2014-SAPV/.

[18] *Collection of supplementary materials*. Embedded Systems Laboratory. Nov. 2017. URL: https://www.ida.liu.se/labs/eslab/alumni/ivanukhov/research/2014-PAPT/.

[19] *Collection of supplementary materials*. Embedded Systems Laboratory. Nov. 2017. URL: https://www.ida.liu.se/labs/eslab/alumni/ivanukhov/research/2015-RAPV/.

[20] *Collection of supplementary materials*. Embedded Systems Laboratory. Nov. 2017. URL: https://www.ida.liu.se/labs/eslab/alumni/ivanukhov/research/2017-PAAI/.

[21] *Collection of supplementary materials*. Embedded Systems Laboratory. Nov. 2017. URL: https://www.ida.liu.se/labs/eslab/alumni/ivanukhov/research/2017-PRU/.

[22] *Collection of supplementary materials*. Embedded Systems Laboratory. Nov. 2017. URL: https://www.ida.liu.se/labs/eslab/alumni/ivanukhov/research/2017-FSPT/.

[23] A. Coskun, T. Rosing, and K. Gross. "Proactive temperature management in MPSoCs." In: *ACM/IEEE International Symposium on Low Power Electronics and Design*. Aug. 2008, pp. 165–170.

[24] A. Coskun, T. Rosing, K. Mihic, G. De Micheli, and Y. Leblebici. "Analysis and optimization of MPSoC reliability." In: *Journal of Low Power Electronics* 2.1 (Apr. 2006), pp. 56–69.

[25] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes. "Energy-efficient resource allocation and provisioning framework for cloud data centers." In: *IEEE Transactions on Network and Service Management* 12.3 (Sept. 2015), pp. 377–391.

[26] A. Das, A. Kumar, and B. Veeravalli. *A survey of lifetime reliability-aware system-level design techniques for embedded multiprocessor systems*. Tech. rep. National University of Singapore, 2014.

[27] A. Das, A. Kumar, and B. Veeravalli. *Reliability-aware platform-based design methodology for energy-efficient multiprocessor systems*. Tech. rep. National University of Singapore, 2014.

[28] A. Das, R. Shafik, G. Merrett, B. Al-Hashimi, A. Kumar, and B. Veeravalli. "Reinforcement learning-based inter- and intra-application thermal optimization for lifetime improvement of multicore systems." In: *Design Automation Conference*. June 2014, 170:1–170:6.

[29] T. Davis. "Algorithm 832: UMFPACK." In: *ACM Transactions on Mathematical Software* 30.2 (June 2004), pp. 196–199.

[30] T. De Mazancourt and D. Gerlic. "The inverse of a block-circulant matrix." In: *IEEE Transactions on Antennas and Propagation* 31.5 (Sept. 1983), pp. 808–810.

[31] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. "A fast and elitist multiobjective genetic algorithm: NSGA-II." In: *IEEE Transactions on Evolutionary Computation* 6.2 (Apr. 2002), pp. 182–197.

[32] J. Díaz, D. García, K. Kim, C.-G. Lee, L. Bello, J. López, S. Min, and O. Mirabella. "Stochastic analysis of periodic real-time systems." In: *IEEE Real-Time Systems Symposium*. Dec. 2002, pp. 289–300.

[33] I. Díaz-Emparanza. "Is a small Monte Carlo analysis a good analysis?" In: *Statistical Papers* 43.4 (Oct. 2002), pp. 567–577.

[34] R. Dick, D. Rhodes, and W. Wolf. "TGFF: Task graphs for free." In: *International Workshop on Hardware/Software Codesign*. Mar. 1998, pp. 97–101.

[35] R. Durrett. *Probability: Theory and examples*. 4th ed. Cambridge University Press, 2010.

[36] M. Eldred, C. Webster, and P. Constantine. "Evaluation of non-intrusive approaches for Wiener–Askey generalized polynomial chaos." In: *AIAA Non-Deterministic Approaches Conference*. 2008.

[37] *Failure mechanisms and models for semiconductor devices*. JEDEC Solid State Technology Association, 2016.

[38] *FFmpeg*. Nov. 2017. URL: `https://ffmpeg.org/`.

[39] F. Firouzi, S. Kiamehr, M. Tahoori, and S. Nassif. "Incorporating the impacts of workload-dependent runtime variations into timing analysis." In: *Design, Automation, and Test in Europe Conference*. Mar. 2013, pp. 1022–1025.

[40] J. Fourier and A. Freeman. *The analytical theory of heat*. Cambridge University Press, 2009.

[41] P. Friedberg, Y. Cao, J. Cain, R. Wang, J. Rabaey, and C. Spanos. "Modeling within-die spatial correlation effects for process-design co-optimization." In: *Symposium on Quality of Electronic Design*. Mar. 2005, pp. 516–521.

[42] A. Gelman, J. Carlin, H. Stern, D. Dunson, A. Vehtari, and D. Rubin. *Bayesian data analysis*. 3rd ed. CRC Press, 2013.

[43] R. Ghanem and P. Spanos. *Stochastic finite element method: A spectral approach*. Springer-Verlag New York, 1991.

[44] P. Ghanta, S. Vrudhula, S. Bhardwaj, and R. Panda. "Stochastic variational analysis of large power grids considering intra-die correlations." In: *Design Automation Conference*. July 2006, pp. 211–216.

[45] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT Press, 2016.

[46] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: Data mining, inference, and prediction*. 2nd ed. Springer-Verlag New York, 2013.

[47] F. Heiss and V. Winschel. "Likelihood approximation by numerical integration on sparse grids." In: *Journal of Econometrics* 144.1 (May 2008), pp. 62–80.

[48] M. Hochbruck and A. Ostermann. "Exponential integrators." In: *Acta Numerica* 19 (May 2010), pp. 209–286.

[49] S. Hochreiter and J. Schmidhuber. "Long short-term memory." In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780.

[50] L. Huang, F. Yuan, and Q. Xu. "Lifetime reliability-aware task allocation and scheduling for MPSoC platforms." In: *Design, Automation, and Test in Europe Conference*. Apr. 2009, pp. 51–56.

[51] P.-Y. Huang, J.-H. Wu, and Y.-M. Lee. "Stochastic thermal simulation considering spatial correlated within-die process variations." In: *Asia and South Pacific Design Automation Conference*. Jan. 2009, pp. 31–36.

[52] W. Huang, K. Sankaranarayanan, K. Skadron, R. Ribando, and M. Stan. "Accurate, pre-RTL temperature-aware design using a parameterized, geometric thermal model." In: *IEEE Transactions on Computers* 57.9 (Sept. 2008), pp. 1277–1288.

[53] S. Ismaeel and A. Miri. "Using ELM techniques to predict data centre VM requests." In: *IEEE International Conference on Cyber Security and Cloud Computing*. Nov. 2015, pp. 80–86.

[54] *ITRS Reports*. International Technology Roadmap for Semiconductors. Nov. 2017. URL: http://www.itrs2.net/.

[55] J. Jakeman and S. Roberts. "Local and dimension adaptive stochastic collocation for uncertainty quantification." In: *Sparse Grids and Applications*. Springer-Verlag Berlin Heidelberg, 2012, pp. 181–203.

[56] S. Janson. *Gaussian Hilbert spaces*. Cambridge University Press, 1997.

[57] S. Joe and F. Kuo. "Constructing Sobol sequences with better two-dimensional projections." In: *SIAM Journal on Scientific Computing* 30.5 (2008), pp. 2635–2654.

[58] D.-C. Juan, Y.-L. Chuang, D. Marculescu, and Y.-W. Chang. "Statistical thermal modeling and optimization considering leakage power variations." In: *Design, Automation, and Test in Europe Conference*. Apr. 2012, pp. 605–610.

[59] D.-C. Juan, S. Garg, and D. Marculescu. "Statistical thermal evaluation and mitigation techniques for 3D chip-multiprocessors in the presence of process variations." In: *Design, Automation, and Test in Europe Conference*. Mar. 2011, pp. 383–388.

[60] D.-C. Juan, S. Garg, and D. Marculescu. "Statistical peak temperature prediction and thermal yield improvement for 3D chip multiprocessors." In: *ACM Transactions on Design Automation of Electronic Systems* 19.4 (Aug. 2014), 39:1–39:23.

[61] S. Kiamehr, F. Firouzi, and M. Tahoori. "Aging-aware timing analysis considering combined effects of NBTI and PBTI." In: *International Symposium on Quality Electronic Design*. Mar. 2013, pp. 53–59.

[62] S. Kiamehr, P. Weckx, M. Tahoori, B. Kaczer, H. Kukner, P. Raghavan, G. Groeseneken, and F. Catthoor. "The impact of process variation and stochastic aging in nanoscale VLSI." In: *IEEE International Reliability Physics Symposium*. Apr. 2016, pp. CR-1-1–CR-1-6.

[63] D. Kingma and J. Ba. "Adam: A method for stochastic optimization." In: *CoRR* (Dec. 2014). arXiv: 1412.6980.

[64] A. Klimke. "Uncertainty modeling using fuzzy arithmetic and sparse grids." PhD thesis. Universität Stuttgart, 2006.

[65] O. Knio and O. Le Maître. "Uncertainty propagation in CFD using polynomial chaos decomposition." In: *Fluid Dynamics Research* 38.9 (Sept. 2006), pp. 616–640.

[66] F. Kreith. *CRC handbook of thermal engineering*. CRC Press, 2000.

[67] P. Kumar and D. Atienza. "Neural network based on-chip thermal simulator." In: *IEEE International Symposium on Circuits and Systems*. May 2010, pp. 1599–1602.

[68] O. Le Maître and O. Knio. *Spectral methods for uncertainty quantification with applications to computational fluid dynamics*. Springer Netherlands, 2010.

[69] Y.-M. Lee and P.-Y. Huang. "An efficient method for analyzing on-chip thermal reliability considering process variations." In: *ACM Transactions on Design Automation of Electronic Systems* 18.3 (July 2013), 41:1–41:32.

[70] H. Li, Z. Lü, and X. Yuan. "Nataf transformation based point estimate method." In: *Chinese Science Bulletin* 53.17 (Sept. 2008), pp. 2586–2592.

[71] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. "Efficient hyperparameter optimization and infinitely many armed bandits." In: *CoRR* (Mar. 2016). arXiv: 1603.06560.

[72] S. Li, J. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi. "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures." In: *IEEE/ACM International Symposium on Microarchitecture*. Dec. 2009, pp. 469–480.

[73] W. Liao, L. He, and K. Lepak. "Temperature and supply voltage aware performance and power modeling at mictoarchitecture level." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24.7 (July 2005), pp. 1042–1053.

[74] P.-L. Liu and A. Der Kiureghian. "Multivariate distribution models with prescribed marginals and covariances." In: *Probabilistic Engineering Mechanics* 1.2 (June 1986), pp. 105–112.

[75] Y. Liu, R. Dick, L. Shang, and H. Yang. "Accurate temperature-dependent integrated circuit leakage power estimation is easy." In: *Design, Automation, and Test in Europe Conference*. Apr. 2007, pp. 1526–1531.

[76] X. Ma and N. Zabaras. "An adaptive hierarchical sparse grid collocation algorithm for the solution of stochastic differential equations." In: *Journal of Computational Physics* 228.8 (May 2009), pp. 3084–3113.

[77] Y. Marzouk and H. Najm. "Dimensionality reduction and polynomial chaos acceleration of Bayesian inference in inverse problems." In: *Journal of Computational Physics* 228.6 (Apr. 2009), pp. 1862–1902.

[78] F. Mesa-Martinez, J. Nayfach-Battilana, and J. Renau. "Power model validation through thermal measurements." In: *ACM/IEEE International Symposium of Computer Architecture*. May 2007, pp. 302–311.

[79] R. Nelsen. *An introduction to copulas*. 2nd ed. Springer-Verlag New York, 2006.

[80] M. Niknafs, I. Ukhov, P. Eles, and Z. Peng. "Two-phase interarrival time prediction for runtime resource management." In: *Euromicro Conference on Digital System Design*. Aug. 2017, pp. 524–528.

[81] M. Niknafs, I. Ukhov, P. Eles, and Z. Peng. "Workload prediction for runtime resource management." In: *IEEE Nordic Circuits and System Conference*. Oct. 2017.

[82] F. Nobile, R. Tempone, and C. Webster. "An anisotropic sparse grid stochastic collocation method for elliptic partial differential equations with random input data." In: *SIAM Journal on Numerical Analysis* 46.5 (May 2008), pp. 2411–2442.

[83] F. Oboril and M. Tahoori. "ExtraTime: Modeling and analysis of wearout due to transistor aging at microarchitecture-level." In: *IEEE/IFIP International Conference on Dependable Systems and Networks*. June 2012, pp. 1–12.

[84] S. Paek, S.-H. Moon, W. Shin, J. Sim, and L.-S. Kim. "PowerField: A transient temperature-to-power technique based on Markov random field theory." In: *Design Automation Conference*. June 2012, pp. 630–635.

[85] S. Pagani, J.-J. Chen, M. Shafique, and J. Henkel. "MatEx: Efficient transient and peak temperature computation for compact thermal models." In: *Design, Automation, and Test in Europe Conference*. Mar. 2015, pp. 1515–1520.

[86] S. Pagani, H. Khdr, W. Munawar, J.-J. Chen, M. Shafique, M. Li, and J. Henkel. "Thermal safe power: Efficient power budgeting for many-core systems in dark silicon." In: *International Conference on Hardware/Software Codesign and System Synthesis*. Oct. 2014, 10:1–10:10.

[87] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical recipes: The art of scientific computing*. 3rd ed. Cambridge University Press, 2007.

[88] S. Quinton, R. Ernst, D. Bertrand, and P. Yomsi. "Challenges and new trends in probabilistic timing analysis." In: *Design, Automation, and Test in Europe Conference*. Mar. 2012, pp. 810–815.

[89] D. Rai, H. Yang, I. Bacivarov, J.-J. Chen, and L. Thiele. "Worst-case temperature analysis for real-time systems." In: *Design, Automation, and Test in Europe Conference*. Mar. 2011, pp. 631–636.

[90] R. Rao. *Linear statistical inference and its applications*. 2nd ed. Wiley-Interscience, 2002.

[91] R. Rao and S. Vrudhula. "Fast and accurate prediction of the steady-state throughput of multicore processors under thermal constraints." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28.10 (Oct. 2009), pp. 1559–1572.

[92] C. Rasmussen and C. Williams. *Gaussian processes for machine learning*. MIT Press, 2006.

[93] S. Reda and S. Nassif. "Analyzing the impact of process variations on parametric measurements: Novel models and applications." In: *Design, Automation, and Test in Europe Conference*. Apr. 2009, pp. 375–380.

[94] C. Reiss, J. Wilkes, and J. Hellerstein. *Google cluster-usage traces: Format and schema*. Tech. rep. Google Inc., Nov. 2011. URL: `https://github.com/google/cluster-data`.

[95] M. Rosenblatt. "Remarks on a multivariate transformation." In: *Annals of Mathematical Statistics* 23.3 (Sept. 1952), pp. 470–472.

[96] L. Santinelli, P. Yomsi, D. Maxim, and L. Cucu-Grosjean. "A component-based framework for modeling and analyzing probabilistic real-time systems." In: *IEEE International Conference on Emerging Technologies And Factory Automation*. Sept. 2011, pp. 1–8.

[97] M. Schmitz, B. Al-Hashimi, and P. Eles. *System-level design techniques for energy-efficient embedded systems*. Springer US, 2004.

[98] A. Schranzhofer, J.-J. Chen, and L. Thiele. "Power-aware mapping of probabilistic applications onto heterogeneous MPSoC platforms." In: *IEEE Real-Time and Embedded Technology and Applications Symposium*. Apr. 2009, pp. 151–160.

[99] R. Shen, N. Mi, S. Tan, Y. Cai, and X. Hong. "Statistical modeling and analysis of chip-level leakage power by spectral stochastic method." In: *Asia and South Pacific Design Automation Conference*. Jan. 2009, pp. 161–166.

[100] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. "Temperature-aware microarchitecture." In: *ACM/IEEE Annual International Symposium on Computer Architecture*. June 2003, pp. 2–13.

[101] S. Smolyak. "Quadrature and interpolation formulas for tensor products of certain classes of functions." In: *Doklady Akademii Nauk SSSR* 148.5 (1963), pp. 1042–1045.

[102] J. Srinivasan, S. Adve, P. Bose, and J. Rivers. "The impact of technology scaling on lifetime reliability." In: *IEEE/IFIP International Conference on Dependable Systems and Networks*. June 2004, pp. 177–186.

[103] A. Srivastava, D. Sylvester, and D. Blaauw. *Statistical analysis and optimization for VLSI: Timing and power*. Springer US, 2010.

[104] J. Stoer and R. Bulirsch. *Introduction to numerical analysis*. 3rd ed. Springer New York, 2002.

[105] B. Tanasa, U. Bordoloi, P. Eles, and Z. Peng. "Probabilistic response time and joint analysis of periodic tasks." In: *Euromicro Conference on Real-Time Systems*. July 2015, pp. 235–246.

[106] *The BSIM4 model*. Berkeley short-channel IGFET model group. Nov. 2017. URL: https://bsim.berkeley.edu/models/bsim4/.

[107] *The NanGate 45nm open cell library*. NanGate. Nov. 2017. URL: http://www.nangate.com/.

[108] *The predictive technology model*. Nanoscale Integration and Modeling Group. Nov. 2017. URL: http://ptm.asu.edu/.

[109] L. Thiele, L. Schor, H. Yang, and I. Bacivarov. "Thermal-aware system analysis and software synthesis for embedded multi-processors." In: *Design Automation Conference*. June 2011, pp. 268–273.

[110] I. Ukhov, M. Bao, P. Eles, and Z. Peng. "Steady-state dynamic temperature analysis and reliability optimization for embedded multiprocessor systems." In: *Design Automation Conference*. June 2012, pp. 197–204. DOI: 10.1145/2228360.2228399.

[111] I. Ukhov, P. Eles, and Z. Peng. "Probabilistic analysis of power and temperature under process variation for electronic system design." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33.6 (June 2014), pp. 931–944. DOI: 10.1109/TCAD.2014.2301672.

[112] I. Ukhov, P. Eles, and Z. Peng. "Temperature-centric reliability analysis and optimization of electronic systems under process variation." In: *IEEE Transactions on Very Large Scale Integration Systems* 23.11 (Nov. 2015), pp. 2417–2430. DOI: 10.1109/TVLSI.2014.2371249.

[113] I. Ukhov, P. Eles, and Z. Peng. "Probabilistic analysis of electronic systems via adaptive hierarchical interpolation." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36.11 (Nov. 2017), pp. 1883–1896. DOI: 10.1109/TCAD.2017.2705117.

[114] I. Ukhov, D. Marculescu, P. Eles, and Z. Peng. *Fast synthesis of power and temperature profiles for the development of data-driven resource managers*. Tech. rep. Linköping University, Sept. 2017.

[115] I. Ukhov, D. Marculescu, P. Eles, and Z. Peng. *Fine-grained long-range prediction of resource usage in computer clusters*. Tech. rep. Linköping University, Sept. 2017.

[116] I. Ukhov, M. Villani, P. Eles, and Z. Peng. "Statistical analysis of process variation based on indirect measurements for electronic system design." In: *Asia and South Pacific Design Automation Conference*. Jan. 2014, pp. 436–442. DOI: 10.1109/ASPDAC.2014.6742930.

[117] S. Vrudhula, J. Wang, and P. Ghanta. "Hermite polynomial based interconnect analysis in the presence of process variations." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25.10 (Oct. 2006), pp. 2001–2011.

[118] Y. Xiang, T. Chantem, R. Dick, S. Hu, and L. Shang. "System-level reliability modeling for MPSoCs." In: *International Conference on Hardware/Software Codesign and System Synthesis*. Oct. 2010, pp. 297–306.

[119] Y. Xie and W.-L. Hung. "Temperature-aware task allocation and scheduling for embedded MPSoC design." In: *Journal of VLSI Signal Processing Systems* 45.3 (Dec. 2006), pp. 177–189.

[120] D. Xiu. *Numerical methods for stochastic computations: A spectral method approach*. Princeton University Press, 2010.

[121] C.-Y. Yang, J.-J. Chen, L. Thiele, and T.-W. Kuo. "Energy-efficient real-time task scheduling with temperature-dependent leakage." In: *Design, Automation, and Test in Europe Conference*. Mar. 2010, pp. 9–14.

[122] W. Zaremba, I. Sutskever, and O. Vinyals. "Recurrent neural network regularization." In: *CoRR* (Sept. 2014). arXiv: 1409.2329.

[123] W. Zhang, X. Li, and R. Rutenbar. "Bayesian virtual probe: Minimizing variation characterization cost for nanoscale IC technologies via Bayesian inference." In: *Design Automation Conference*. June 2010, pp. 262–267.

[124] D. Zhu, H. Aydin, and J.-J. Chen. "Optimistic reliability aware energy management for real-time tasks with probabilistic execution times." In: *IEEE Real-Time Systems Symposium*. Nov. 2008, pp. 313–322.