Probabilistic Analysis of Electronic Systems via Adaptive Hierarchical Interpolation

Ivan Ukhov, Petru Eles, *Member, IEEE*, and Zebo Peng, *Senior Member, IEEE*

***Abstract*—We present 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 the framework general and suitable for studying various metrics that are of interest to the designer. Examples of such metrics include the end-to-end delay, total energy consumption, and maximum temperature of the system under consideration. The framework delivers a light generative representation that allows for a straightforward, computationally efficient calculation of the probability distribution and accompanying statistics of the metric at hand. Our technique is illustrated by considering a number of uncertainty-quantification problems and comparing the corresponding results with exhaustive simulations.**

***Index Terms*—Adaptive interpolation, computer simulation, electronic system, probabilistic analysis, sparse grid, statistical dependence, uncertainty quantification.**

I. Introduction

PROBABILISTIC ANALYSIS of electronic systems is an extensive and diverse area, which is expanding with an accelerating pace. The rapid growth is instigated by the fact that electronic systems naturally become more sophisticated and refined, and that they penetrate deeper into everyday life. Therefore, the impact of uncertainty inevitably becomes more prominent and entails more severe consequences, necessitating an adequate treatment. Consequently, the designer of an electronic system is obliged to account for the presence of uncertainty in order to produce an efficient and reliable product.

In order to account for uncertainty, one has to quantify it first. In this setting, one is usually interested in evaluating a certain metric whose complete knowledge would be highly profitable for the design at hand but cannot be attained since the metric involves a number of parameters that are inherently uncertain at design time. To give a concrete example, such a metric could be the maximum temperature of an electronic system whose thermal behavior depends on the runtime workload.

Uncertainty quantification is a broad area. The techniques for uncertainty quantification can deliver radically different pieces of information about the metric under consideration. In this paper, we are interested in probability distributions rather than, for instance, corner cases. Designing for the worst case leads often to a poor solution as the system under consideration might easily end up being too conservative, overdesigned [1].

When it comes to the estimation of probability distributions and to uncertainty quantification in general, sampling methods are of great use. The classical Monte Carlo (MC) sampling, quasi-MC sampling, and Latin hypercube sampling are examples of such methods [2]. Compared to other techniques for probabilistic analysis, these methods are straightforward to apply. 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.

The major problem with sampling techniques, however, is in sampling: one should be able to obtain sufficient many realizations of the metric of interest in order to accurately estimate the needed statistics about that metric [3]. When the subject under analysis is expensive to evaluate, sampling methods are rendered slow and often unfeasible.

We propose a design-time system-level framework for the analysis of electronic systems that are dependent on uncertain parameters. Similar to sampling methods, our technique treats the system at hand as a "black box" and, therefore, is straightforward to apply since no handcrafting is required, and existing codes need no change. Consequently, the metrics that the framework is able to tackle are diverse. Examples include those metrics concerned with timing-, power-, and temperature-related characteristics of elaborate applications running on heterogeneous multiprocessor platforms.

In contrast to sampling methods, our technique explores and exploits the nature of the problem—that is, the way the metric depends on the uncertain parameters—by exercising the aforementioned "black box" at a set of points chosen adaptively. The adaptivity that our framework leverages is hybrid [4]: it tries to pick up both global (that is, on the level of individual dimensions [5]) and, more importantly, local (that is, on the level on individual points [6]) variations. This means that the framework is able to benefit from many particularities that might be present in the stochastic space, that is, the space of the uncertain parameters. The adaptivity is the capital feature of our technique, and we would like to elaborate on it now.

The uncertainties present in electronic systems originate from both the physical world and the computer world. An example of a physical source of uncertainty is process variation [7], which is a side effect of contemporary fabrication processes. Process variation has been central for many lines of research [8], [9], [10], [11], [12]. An example of a digital source of uncertainty (the computer world) is workload. To elaborate, the characteristics of the codes running on modern devices change from one activation to another depending on the environment and input data. This source of uncertainty has not been deprived of attention either, especially in the real-time community [1], [13], [14], [15]. Regardless of the origin, such phenomena as the ones mentioned above render the behavior of an electronic system nondeterministic to the designer.

Due to its nature, the variability coming from the physical world is typically smooth, well behaved. In such cases, uncertainty quantification based on polynomial chaos (PC) expansions [16] and other approximation techniques making use of global polynomials generally work well, as in [8], [10], [11], [12]. On the other hand, the variability coming from the computer world has often steep gradients and favors

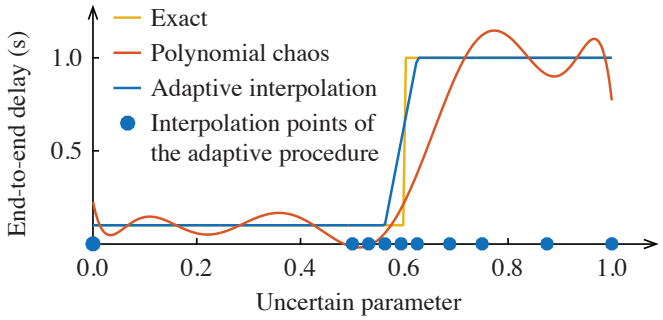

Figure 1. An illustration of the accuracy of polynomial chaos expansions and adaptive interpolation applied to a nonsmooth metric of interest.

nondifferentiability and even discontinuity. In such cases, PC expansions and similar techniques fail: they require extremely many evaluations of the desired metric in order to deliver an acceptable level of accuracy and, hence, are 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 further 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 metric 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 random variable u uniformly distributed on $[0, 1]$: 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 yellow line in Fig. 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 [16]. The orange line in Fig. 1 shows a ninth-order PC expansion, which uses 10 points. 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. No matter how many points are used in the construction of a polynomial, the oscillations will never go away completely.

Let us now see how the framework proposed in this paper solves the same problem. For the purpose of the experiment, our technique is constrained to make use of as many points as the PC expansion did. The result is the blue curve in Fig. 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. One can note that the adaptive procedure started to concentrate interpolation points at the jump and left the insipid regions on both sides of the jump with no particular attention. Having constructed such a representation, one can proceed to the calculation of the probability distribution of the metric, which, in general, is done via sampling followed by such techniques as kernel density estimation. The crucial point to note is that this follow-up sampling does not involve the original system in any way, which implies that it costs practically nothing in terms of the computation time.

The example discussed 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 of the costs associated with probabilistic analysis of the metric under consideration, as measured by the number of times the metric needs to be evaluated in order to achieve a certain accuracy level. The magnitude of the reduction depends on the problem, and it can be substantial when the problem is well disposed to adaptation.

The remainder of the paper is organized as follows. Section II provides an overview of the prior work and summarizes our contribution. In Sec. III, the problem that we consider is formulated, and our solution to the problem is outlined. The framework proposed to tackle the problem is presented in Sec. IV, V, and VI. The experimental results are given in Sec. VII. Finally, Sec. VIII concludes the paper.

II. Prior Work and Our Contribution

In this section, we elaborate on the prior work (Sec. II-A) and the contribution of this paper (Sec. II-B).

A. Prior Work

Sampling methods would be a reasonable solution to probabilistic analysis of electronic systems if electronic systems were inexpensive (with respect to the computation time) to simulate. In order to eliminate or reduce the costs associated with direct sampling, a number of techniques have been introduced.

Let us first discuss physical sources of uncertainty and, more concretely, process variation as it has been extensively studied. Circuit-level timing and power analyses under process variation are undertaken in [8] by means of polynomial chaos (PC) expansions [16]. The work in [9] models static steady-state temperature and accounts for process variation by leveraging the linearity of Gaussian distributions and time-invariant systems. A stochastic collocation [16] approach to static steady-state temperature analysis is given in [10], which relies on global interpolation using Newton polynomials. In [11], transient temperature analysis is considered, and process variation is addressed via PC expansions. The machinery of PC expansions is also utilized in [12] in order to model dynamic steady-state temperature [17] and to enhance reliability models.

Let us now turn to digital sources of uncertainty. In this context, timing analysis has drawn the major attention [1]. A seminal work on response time analysis of periodic tasks with random execution times on uniprocessors is reported in [13]. A novel analytical solution to this problem is given in [15], which makes milder assumptions and allows for addressing larger, previously unsolvable problems. The framework proposed in [14] facilitates task scheduling by providing probabilistic bounds on the resource given to a task flow and the resource needed by that task flow; the approach is based on real-time calculus and is applicable to electronic systems.

Studying the literature on probabilistic analysis of electronic systems, one can note a pronounced trend: the generality

and straightforwardness of sampling methods tend to be lost. To elaborate, a technique typically: 1) requires restrictive assumptions to be fulfilled such as the absence of correlations, 2) is tailored to one concrete metric such as the response time, and 3) requires substantial effort to be deployed.

However, one should keep in mind what is practical. First of all, 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 has been developed a robust simulator evaluating the metric at hand for the deterministic scenario. Switching to probabilistic analysis based on analytical approaches means discarding this battle-tested code and implementing something else from scratch, which is wasteful and not desirable.

Some of the techniques listed earlier in this section, in fact, preserve the generality and straightforwardness of sampling methods. An example is the uncertainty analysis presented in [12]. The reason is that the construction of PC expansions in [12] is undertaken by means of so-called nonintrusive spectral projections [16], which do not need to look inside the "black box," similar to sampling methods. However, as motivated in Sec. I, nonsmoothness is a serious problem for global approximation based on polynomials. The convergence of PC expansions, for instance, deteriorates substantially in such cases, requiring partitioning of the stochastic space in order to alleviate the problem. Therefore, it is not straightforward to apply such techniques as the one given in [12] in the context of digital sources of uncertainty exhibiting nonsmoothness.

To conclude, the available techniques for probabilistic analysis of electronic systems are restricted in use. Flexible, capable, and easy-to-deploy frameworks are needed.

B. Our Contribution

Our work brings the following major contribution: we develop an efficient framework for probabilistic analysis of electronic systems that is straightforward to use and is applicable to a wide range of uncertainty-quantification problems.

The usage of our framework is streamlined because it has the same low entrance requirements as sampling techniques: one only has to be able to evaluate the metric given a set of deterministic parameters. Moreover, the framework can be utilized in scenarios with limited knowledge of the joint probability distribution of the uncertain parameters, which are common in practice (to be elaborated on in Sec. IV-A).

The scope of our framework is wide because the framework features a powerful approximation engine. We make use of the hierarchical interpolation with hybrid adaptivity developed in [4], [5], [6], which enables us to tackle diverse design problems while keeping the associated computation costs low.

To the best of our knowledge, our framework is the first one to address systematically and efficiently nonsmooth problems in the context of probabilistic analysis of electronic systems. The importance of the characteristic is motivated in Sec. I.

In addition to the aforementioned contribution, we open-source our implementation [18]. The code base also includes the whole experimental setup described in Sec. VII.

III. Problem Formulation and Our Solution

After introducing a number of definitions (Sec. III-A), this section formulates the problem that we consider (Sec. III-B) and outlines our solution (Sec. III-C). It also contains an example illustrating the solution process (Sec. III-D).

A. Preliminaries

Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space where Ω is a set of outcomes, $\mathcal{F} \subseteq 2^\Omega$ is a σ-algebra, and $\mathbb{P} : \mathcal{F} \to [0,1]$ is a probability measure [19]. A random variable ξ defined on $(\Omega, \mathcal{F}, \mathbb{P})$ is an $\mathcal{F}$-measurable function $\xi : \Omega \to \mathbb{R}$. A random variable is uniquely characterized by its distribution function defined by $F_\xi(z) = \mathbb{P}(\{\omega \in \Omega : \xi(\omega) \leq z\})$, written as $\xi \sim F_\xi$. The expected value and variance of ξ are given by

$$\mathbb{E}\,\xi = \int_\Omega \xi(\omega)\, d\mathbb{P}(\omega) = \int_{\mathbb{R}} z\, dF_\xi(z) \quad \text{and} \tag{1}$$

$$\mathbb{V}\mathrm{ar}\,\xi = \mathbb{E}\,\xi^2 - (\mathbb{E}\,\xi)^2, \ \text{respectively.} \tag{2}$$

A random vector $\boldsymbol{\xi} = (\xi_i)_{i=1}^n$ is a vector whose elements are random variables. A random vector is fully characterized by its distribution function $F_{\boldsymbol{\xi}}$, written as $\boldsymbol{\xi} \sim F_{\boldsymbol{\xi}}$. This function is referred to as a joint or multivariate distribution function, emphasizing the fact that the variables work together.

An n-variate distribution can be expressed as a set of n marginal (univariate) distributions and an n-dimensional copula [20]. The copula is a uniform distribution function on $[0,1]^n$ (referred to as the n-dimensional unit hypercube) that captures the dependencies between the n individual variables.

B. 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, and the application is a collection of interdependent tasks.

The designer is interested in evaluating a metric g that characterizes the electronic system under consideration from a certain perspective. Examples of g include the execution delay of the application or a task, energy consumption of the platform or a processing element, and maximum temperature of the platform or a processing element.

The metric g depends on a set of parameters $\mathbf{u}$ that are uncertain at the design stage. Examples of $\mathbf{u}$ include the amount of data the application needs to process, execution times of the tasks, and properties of the environment.

The parameters $\mathbf{u}$ are given as a random vector $\mathbf{u} = (u_i)_{i=1}^{n_u}$ with an arbitrary but known distribution $F_{\mathbf{u}}$. The dependency of g on $\mathbf{u}$, written as $g(\mathbf{u})$, implies that g is random to the designer. For a given outcome of $\mathbf{u}$, however, the evaluation of g is purely deterministic. This operation is typically undertaken by an adequate simulator of the system at hand, and it is assumed to be doable but computationally expensive.

Our objective is to develop an efficient framework for estimating the probability distribution of the metric of interest g dependent on the uncertain parameters $\mathbf{u}$. The framework is required to be able to handle nondifferentiable and even discontinuous dependencies between g and $\mathbf{u}$ as they constitute an important class of problems for electronic-system design.

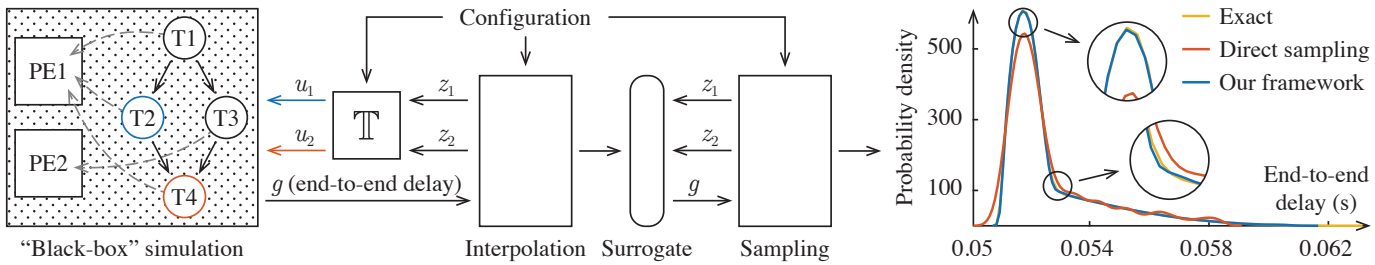

Figure 2. The proposed framework applied to the end-to-end delay of an application whose two out of four tasks have random execution times.

C. Our Solution

As noted earlier, making use of a sampling method is a compelling approach to uncertainty quantification. We would readily apply such a method to study our metric g if only evaluating g had a small cost, which it does not.

Our solution to the above predicament is to construct a light representation of the heavy g and study this representation instead of g. The surrogate that we build is based on adaptive interpolation: 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 mediating between the collected values of g. The benefit of this approach is in the number of invocations of the metric g: only a few evaluations of g are needed, and the rest of our probabilistic analysis is powered by the constructed interpolant, which, in contrast to g, has a negligible cost.

Let us delineate the steps involved in the solution process. Recall that g is parameterized by the uncertain parameters $\mathbf{u}$, and these variables are the only source of randomness. 1) The metric g is reparameterized in terms of an auxiliary random vector $\mathbf{z}$ extracted from $\mathbf{u}$; the necessity of this stage will become clear later on. 2) An interpolant of g is constructed by considering g as a deterministic function of $\mathbf{z}$ and evaluating g at a small set of carefully chosen points. 3) The probability distribution of g is then estimated by applying an arbitrary sampling method to the constructed interpolant of g.

The first two of the above steps should be undertaken with a great care as interpolation of multivariate functions is a challenging task. This aspect will be discussed in detail in Sec. IV and Sec. V. However, before we proceed to those sections, we would like to give an illustrative example.

D. Illustrative Example

In this section, we apply our framework to a small problem in order to get a better understanding of the workflow of the framework. A detailed description of our experimental setup is given in Sec. VII-A; here we give only the bare minimum.

The addressed problem is depicted in Fig. 2. We consider a platform with two processing elements, `PE1` and `PE2`, and an application with four tasks, `T1`–`T4`. The data dependencies between `T1`–`T4` and their mapping onto `PE1` and `PE2` can be seen in Fig. 2. The metric g is the end-to-end delay of the application. The uncertain parameters $\mathbf{u}$ are the execution times of `T2` and `T4` denoted by u_1 and u_2, respectively.

The leftmost box in Fig. 2 represents a simulator of the system at hand, and it could involve such tools as Sniper [21]. It takes an assignment of the execution times of `T2` and `T3`, u_1 and u_2, and outputs the calculated end-to-end delay g. The second box corresponds to the reparameterization mentioned in Sec. III-C (to be discussed in Sec. IV-A). It converts the auxiliary variables z_1 and z_2 into u_1 and u_2 in accordance with u_1 and u_2's joint distribution. The third box is our interpolation engine (to be discussed in Sec. V). Using a number of strategic invocations of the simulator, the interpolation engine yields a light surrogate for the simulator; the surrogate corresponds to the slim box with rounded corners. Having obtained such a surrogate, one proceeds to sampling extensively the surrogate via a sampling method of choice (the rightmost box). The surrogate takes z_1 and z_2 and returns an approximation of g at that point. Recall that the computation cost of this extensive sampling is negligible as g is not involved. The samples are then used to compute an estimate of the distribution of g.

In the graph on the right-hand side of Fig. 2, the blue line shows the probability density function of g computed by applying kernel density estimation to the samples obtained from our surrogate. The yellow line (barely visible behind the blue line) shows the true density of g; its calculation is explained in Sec. VII. It can be seen that our solution closely matches the exact one. In addition, the orange line shows the estimation that one would get if one sampled g directly 156 times and used only those samples in order to calculate the density of g. We see that, for the same budget of simulations, the solution delivered by our framework is substantially closer to the true one than the one delivered by naïve sampling.

At this point, we are ready to present to the proposed framework. We begin by elaborating on the modeling of uncertain parameters and metrics of interest. We shall then proceed to the interpolation engine (Sec. V).

IV. Modeling

The agenda for this section is as follows. In Sec. IV-A, the uncertain parameters $\mathbf{u}$ are transformed into a form suitable for the subsequent calculations. This stage is an essential part of our framework, and it is denoted by $\mathbb{T}$ in Fig. 2. The rest of the subsections, Sec. IV-B–IV-D, serve a strictly illustrative purpose. They exemplify the leftmost box in Fig. 2 in order to give the reader a better intuition about the utility of the framework. The subsections introduce a number of models and a number of metrics g; however, it should be well understood that the essence of g is problem specific. In practice, g stands for an adequate simulator of the system under consideration. The modeling capabilities of this simulator are naturally inherited by the proposed framework.

A. Uncertainty Parameters

The foremost step of our framework is to change the parameterization of the problem from the random vector $\mathbf{u} = (u_i)_{i=1}^{n_u} \sim F_{\mathbf{u}}$ to an auxiliary random vector $\mathbf{z} = (z_i)_{i=1}^{n_z} \sim F_{\mathbf{z}}$ such that: 1) the support of $F_{\mathbf{z}}$ is the unit hypercube $[0,1]^{n_z}$, and 2) $n_z \leq n_u$ has the smallest value needed to retain the desired level of accuracy. The first is standardization, which is done primarily for convenience. The second is model-order reduction, which identifies and eliminates excessive complexity and, hence, speeds up the solution process. The reduction is possible whenever there are dependencies between $(u_i)_{i=1}^{n_u}$, in which case one can find such $(z_i)_{i=1}^{n_z}$, $n_z < n_u$, that each u_i can be recovered from $(z_i)_{i=1}^{n_z}$. We shall denote the overall transformation by $\mathbf{u} = \mathbb{T}(\mathbf{z})$ where

$$\mathbb{T} : \mathbb{R}^{n_u} \to [0,1]^{n_z}. \tag{3}$$

For any point $\mathbf{z} \in [0,1]^{n_z}$, we are now able to compute the corresponding $\mathbf{u}$ and, consequently, the metric g as $(g \circ \mathbb{T})(\mathbf{z}) = g(\mathbb{T}(\mathbf{z})) = g(\mathbf{u})$; recall Sec. III-B and see Fig. 2.

Let us consider an example of $\mathbb{T}$ in order to understand the concept better. To this end, we begin by assuming that the distribution of $\mathbf{u} = (u_i)_{i=1}^{n_u}$, $F_{\mathbf{u}}$, is given as a set of marginal distribution functions $\{F_{u_i}\}_{i=1}^{n_u}$ and a copula [20] (see also Sec. III-A). Furthermore, the copula is assumed to be a Gaussian copula whose correlation matrix is $\mathbf{R} \in \mathbb{R}^{n_u \times n_u}$.

Remark 1. *A set of marginals and a copula entirely characterize the joint distribution of* $\mathbf{u}$*, that is,* $F_{\mathbf{u}}$*. However, we consider this distribution as an approximation rather than as the true one. The knowledge of the true joint would be an impractical assumption to make. A more realistic assumption is the availability of the marginals and correlation matrix of* $\mathbf{u}$*. In general, these two pieces are not sufficient to recover the joint of* $\mathbf{u}$*; however, the joint can be approximated well by accompanying the available marginals by a Gaussian copula constructed based on the available correlation matrix; see [22] and also [11]. Hence, a set of marginals and a Gaussian copula are practical inputs to probabilistic analysis.*

The number of variables, which is so far n_u, has a significant impact on the complexity of the problem at hand. Therefore, an important component of our framework is model-order reduction, which we shall base on the discrete Karhunen–Loève decomposition, also known as the principal component analysis. We proceed as follows. Since any correlation matrix is real and symmetric, $\mathbf{R}$ admits the eigendecomposition: $\mathbf{R} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$ where $\mathbf{V} \in \mathbb{R}^{n_u \times n_u}$ is an orthogonal matrix whose columns are the eigenvectors of $\mathbf{R}$, and $\mathbf{\Lambda} = \operatorname{diag}(\lambda_i)_{i=1}^{n_u}$ is a diagonal matrix whose diagonal elements are the eigenvalues of $\mathbf{R}$. The eigenvalues $(\lambda_i)_{i=1}^{n_u}$ correspond to the variances of the corresponding components revealed by the decomposition. The model-order reduction boils down to selecting those major components whose cumulative contributions to the total variance is above a certain threshold. Formally, assuming that $(\lambda_i)_{i=1}^{n_u}$ are sorted in the descending order and given a threshold $\eta \in (0,1]$ specifying the fraction of the total variance to be preserved, we identify the smallest n_z such that

$$\frac{\sum_{i=1}^{n_z} \lambda_i}{\sum_{i=1}^{n_u} \lambda_i} \geq \eta. \tag{4}$$

Denote by $\tilde{\mathbf{V}} \in \mathbb{R}^{n_u \times n_z}$ and $\tilde{\mathbf{\Lambda}} \in \mathbb{R}^{n_z \times n_z}$ the matrices obtained by truncating $\mathbf{V}$ and $\mathbf{\Lambda}$, respectively, to preserve only the first n_z components where n_z is as shown above.

Now, the transformation $\mathbb{T}$ in (3) is

$$\mathbf{u} = F_{\mathbf{u}}^{-1}\left(\Phi\left(\tilde{\mathbf{V}}\tilde{\mathbf{\Lambda}}^{\frac{1}{2}}\,\Phi^{-1}(\mathbf{z})\right)\right) \tag{5}$$

where the random variables $\mathbf{z} = (z_i)_{i=1}^{n_z}$ are independent and uniformly distributed on $[0,1]^{n_z}$; Φ and Φ^{-1} are the distribution function of the standard Gaussian distribution and its inverse, respectively, which are applied elementwise; and $F_{\mathbf{u}}^{-1} = F_{u_1}^{-1} \times \cdots \times F_{u_{n_z}}^{-1}$ is the Cartesian product of the inverse marginal distributions of $\mathbf{u}$, which are applied to the corresponding element of the vector yielded by Φ. In the absence of correlations, (5) is simply $\mathbf{u} = F_{\mathbf{u}}^{-1}(\mathbf{z})$, and no model-order reduction is possible ($n_u = n_z$).

To summarize, we have found such a transformation $\mathbb{T}$ and the corresponding random vector $\mathbf{z} \sim F_{\mathbf{z}}$ that: 1) $F_{\mathbf{z}}$ is supported by $[0,1]^{n_z}$, and 2) $\mathbf{z}$ has the smallest number of dimensions n_z needed to preserve η portion of the variance. Let us emphasize that this $\mathbb{T}$ is an example; the framework works with any $\mathbb{T}$ that yields $\mathbf{z} \sim F_{\mathbf{z}}$ for some n_z.

B. Application Timing

Suppose the application is given as a directed acyclic graph. The vertices represent tasks, and the edges data dependency between these tasks. Suppose further that a static cyclic scheduling policy is utilized. Note, however, these assumptions are orthogonal to our framework: the framework can be applied to any application model and any scheduling policy.

Each task has a start and a finish time. For task i, denote these two time moments by b_i and d_i, respectively, and let $\mathbf{b} = (b_i)_{i=1}^{n_t}$ and $\mathbf{d} = (d_i)_{i=1}^{n_t}$. Other timing characteristics of the application can be derived from $(\mathbf{b}, \mathbf{d})$. An example is the end-to-end delay, which is the difference between the finish time of the latest task and the start time of the earliest task:

$$\text{End-to-end delay} = \max_{i=1}^{n_t} d_i - \min_{i=1}^{n_t} b_i. \tag{6}$$

Suppose the execution times of the tasks depend on $\mathbf{u}$ (see Sec. III-B). Then the tuple $(\mathbf{b}, \mathbf{d})$ depends on $\mathbf{u}$. Then the end-to-end delay given in (6) depends on $\mathbf{u}$ and is a potential metric g; it is used in Fig. 2. Note that this g is nondifferentiable as the $\max$ and $\min$ functions are such. Hence, g is nonsmooth, which renders PC expansions and similar techniques inadequate for this problem, as illustrated in Sec. I.

Remark 2. *In general, the behavior of* g *with respect to continuity, differentiability, and smoothness cannot be inferred from the behavior of* $\mathbf{u}$*. Even when the parameters are perfectly behaved,* g *can still and likely will exhibit nondifferentiability or even discontinuity, which depends on how* g *works internally. For example, as shown in [15], even if execution times of tasks are continuous, due to the actual scheduling policy, end-to-end delays are very often discontinuous.*

C. Power Consumption

Denote the number of processing elements present on the platform by n_π. Let the dynamic power consumed by task j when running on processing element i be fixed during the execution of the task and denote this dynamic power

by p_{ij}^{d}. The fact that p_{ij}^{d} is constant might seem restrictive. However, one should keep in mind that it is an example. Our framework does not have such a restriction. Even in this simple model, the modeling accuracy can be substantially improved by representing large tasks as sequences of smaller tasks.

Let the vector $\mathbf{p}(t) = (p_i(t))_{i=1}^{n_\pi}$ capture the total power consumption of the system at time t. This vector is related to the dynamic power introduced above as follows:

$$p_i(t) = \sum_{j=1}^{n_t} p_{ij}^{\mathrm{d}}\, \delta_{ij}(t) + p_i^{\mathrm{s}}(t), \quad \text{for } i = 1, \ldots, n_\pi, \tag{7}$$

where $\delta_{ij}(t)$ is an indicator function (outputs either zero or one) of the event that processing element i executes task j at time t, and $p_i^{\mathrm{s}}(t)$ is the static power consumed by processing element j at time t. The last component depends on time because the leakage power and temperature are interdependent [23], and temperature changes over time (see the next subsection).

Given a set of n_t points on the timeline $\{t_i\}_{i=1}^{n_t}$, (7) can be used to construct a power profile of the system as follows:

$$\mathbf{P} = (p_i(t_j))_{i=1,j=1}^{n_\pi,n_t} \in \mathbb{R}^{n_\pi \times n_t}.$$

The above is a matrix where row i captures the power consumed by processing element i at the n_t time moments.

The total energy consumed by the system during an application run can be computed by integrating (7) over the time span of the application—which is demarcated by the minuend and subtrahend in (6)—and the corresponding integral can be estimated using the power profile as follows:

$$\text{Total energy} = \sum_{i=1}^{n_\pi} \int p_i(t)\, dt \approx \sum_{i=1}^{n_\pi} \sum_{j=1}^{n_t} p_i(t_j)\, \Delta t_j \tag{8}$$

where Δt_j is either $t_j - t_{j-1}$ or $t_{j+1} - t_j$, depending on how power values are encoded in $\mathbf{P}$. The assumption that (8) is based on is that each Δt_i is sufficiently small so that the power consumed within the interval does not change significantly.

Since the tuple $(\mathbf{b}, \mathbf{d})$ depends on $\mathbf{u}$, the power consumption of the system depends on $\mathbf{u}$ too. Consequently, the total energy given in (8) depends on $\mathbf{u}$ and is a candidate for g. Note that Remark 2 applies in this context to the full extent.

D. Heat Dissipation

Based on the specification of the platform including its thermal package, an equivalent thermal RC circuit is constructed [24]. The circuit comprises n_{n} thermal nodes, and its structure depends on the intended level of granularity, which impacts the resulting accuracy. For clarity, we assume that each processing element is mapped onto one corresponding node, and the thermal package is represented as a set of additional nodes.

The thermal dynamics of the system are modeled using the following system of differential-algebraic equations [11], [17]:

$$\begin{cases} \mathbf{C}\dfrac{d\mathbf{s}(t)}{dt} + \mathbf{G}\mathbf{s}(t) = \mathbf{M}\mathbf{p}(t) & \text{(9a)} \\ \mathbf{q}(t) = \mathbf{M}^T\mathbf{s}(t) + \mathbf{q}_{\text{amb}} & \text{(9b)} \end{cases}$$

The coefficients $\mathbf{C} \in \mathbb{R}^{n_{\mathrm{n}} \times n_{\mathrm{n}}}$ and $\mathbf{G} \in \mathbb{R}^{n_{\mathrm{n}} \times n_{\mathrm{n}}}$ are a diagonal matrix of thermal capacitance and a symmetric, positive-definite matrix of thermal conductance, respectively. The vectors $\mathbf{p}(t) \in \mathbb{R}^{n_\pi}$, $\mathbf{q}(t) \in \mathbb{R}^{n_\pi}$, and $\mathbf{s}(t) \in \mathbb{R}^{n_{\mathrm{n}}}$ correspond the system's power, temperature, and internal state at time t, respectively. The vector $\mathbf{q}_{\text{amb}} \in \mathbb{R}^{n_\pi}$ contains the ambient temperature. The matrix $\mathbf{M} \in \mathbb{R}^{n_{\mathrm{n}} \times n_\pi}$ is a mapping that distributes the power consumption of the processing elements across the thermal nodes; without loss of generality, $\mathbf{M}$ is a rectangular diagonal matrix whose diagonal elements are equal to one.

Given a set of n_t points on the timeline $\{t_i\}_{i=1}^{n_t}$, (9) can be used to compute a temperature profile of the system as follows:

$$\mathbf{Q} = (q_i(t_j))_{i=1,j=1}^{n_\pi,n_t} \in \mathbb{R}^{n_\pi \times n_t}.$$

Then the maximum temperature of the system can be estimated using the temperature profile as follows:

$$\text{Max temperature} = \max_{i=1}^{n_\pi} \sup_t\, q_i(t) \approx \max_{i=1}^{n_\pi} \max_{j=1}^{n_t}\, q_i(t_j). \tag{10}$$

Since the power consumption of the system is affected by $\mathbf{u}$ (see Sec. IV-C), the system's temperature is affected by $\mathbf{u}$ as well. Therefore, the temperature in (10) can be considered as a metric g. Note that, due to the maximization involved, the metric is nondifferentiable and, hence, cannot be adequately addressed using polynomial approximations, specially taking into account the concern in Remark 2.

To sum up, we have discussed the transformation that needs to be applied to $\mathbf{u}$ prior to the interpolation of g. We have also covered three aspects of electronic systems, namely, timing, power, and temperature, and introduced a number of metrics associated with them; we shall come back to these metrics in the section on experimental results, Sec. VII.

V. Interpolation

In this section, we present the algorithm that constitutes the core of the proposed framework for probabilistic analysis of electronic systems. It corresponds to the third box from the left in Fig. 2. The algorithm features a sparse-grid structure, hierarchical construction, and hybrid adaptivity. The benefits of these features are interconnected and can be summarized as follows: the ability to efficiently tackle multidimensional problems, the ability to perform gradual refinement of the approximation with a natural error control, and the ability to make the refinement fine-grained and, therefore, gain further efficiency. The mathematics presented in Sec. V-A–V-E is a distilled version of the one developed in [4], [5], [6].

Let f be a function that we would like to approximate; the connection between f and g will be explained in Sec. VI. The function is assumed to be in $\mathcal{C}([0,1]^{n_{\mathrm{d}}})$, the space of continuous functions in the unit hypercube $[0,1]^{n_{\mathrm{d}}}$. The assumption is a formality and does not impose any restrictions in practice.

A. Tensor Product

In one dimension ($n_{\mathrm{d}} = 1$), f is approximated by virtue of the following interpolation formula:

$$\mathcal{Q}_i(f) = \sum_{j \in \mathcal{J}_i} f(x_{ij})\, e_{ij} \tag{11}$$

where $i \geq 0$ signifies the level of interpolation; $\mathcal{X}_i = \{x_{ij}\}_{j \in \mathcal{J}_i} \subset [0,1]$ are the collocation nodes; $\mathcal{E}_i = \{e_{ij}\}_{j \in \mathcal{J}_i} \subset \mathcal{C}([0,1])$ are the basis functions; and $\mathcal{J}_i = \{j-1\}_{j=1}^{n_i}$ is an index set enumerating (starting from zero) the nodes and functions of level i. The subscript $j \in \mathcal{J}_i$ is referred to as the order of a node or function. The choice of $\mathcal{X}_i$ and $\mathcal{E}_i$ is important and will be discussed thoroughly later on.

In multiple dimensions ($n_\mathrm{d} > 1$), f is approximated by the tensor product of n_d one-dimensional interpolants:

$$\mathcal{Q}_\mathbf{i}(f) = \left(\bigotimes_{k=1}^{n_\mathrm{d}} \mathcal{Q}_{i_k}\right)(f) = \sum_{\mathbf{j}\in\mathcal{J}_\mathbf{i}} f(\mathbf{x_{ij}})\, e_\mathbf{ij} \tag{12}$$

where $\mathbf{i} = (i_k)_{k=1}^{n_\mathrm{d}}$ and $\mathbf{j} = (j_k)_{k=1}^{n_\mathrm{d}}$ are (multi-)indices specifying levels and orders, respectively, for each of the dimensions, and $\mathcal{J}_\mathbf{i} = \mathcal{J}_{i_1} \times \cdots \times \mathcal{J}_{i_{n_\mathrm{d}}}$ is an index set obtained by computing the Cartesian product of one-dimensional index sets. In the above formula,

$$\begin{aligned}\mathcal{X}_\mathbf{i} &= \mathcal{X}_{i_1} \times \cdots \times \mathcal{X}_{i_{n_\mathrm{d}}} \\ &= \{\mathbf{x_{ij}} = (x_{i_k j_k})_{k=1}^{n_\mathrm{d}}\}_{\mathbf{j}\in\mathcal{J}_\mathbf{i}} \subset [0,1]^{n_\mathrm{d}}\end{aligned} \tag{13}$$

and

$$\mathcal{E}_\mathbf{i} = \bigotimes_{k=1}^{n_\mathrm{d}} \mathcal{E}_{i_k} = \left\{ e_\mathbf{ij} = \bigotimes_{k=1}^{n_\mathrm{d}} e_{i_k j_k} \right\}_{\mathbf{j}\in\mathcal{J}_\mathbf{i}} \subset \mathcal{C}([0,1]^{n_\mathrm{d}}) \tag{14}$$

are the collocation nodes and basis functions, respectively, corresponding to index $\mathbf{i}$. In (14), for any $\mathbf{x} \in [0,1]^{n_\mathrm{d}}$,

$$e_\mathbf{ij}(\mathbf{x}) = \left(\bigotimes_{k=1}^{n_\mathrm{d}} e_{i_k j_k}\right)(\mathbf{x}) = \prod_{k=1}^{n_\mathrm{d}} e_{i_k j_k}(x_k). \tag{15}$$

Finally, the cardinality of $\mathcal{J}_\mathbf{i}$ is as follows:

$$n_\mathbf{i} = |\mathcal{J}_\mathbf{i}| = \prod_{k=1}^{n_\mathrm{d}} |\mathcal{J}_{i_k}| = \prod_{k=1}^{n_\mathrm{d}} n_{i_k}. \tag{16}$$

Equation (16) elucidates the prohibitive expense of the tensor-product construction shown in (12) for multidimensional problems: the number of nodes grows exponentially as n_d increases. However, (12) serves well as a building block for more efficient algorithms, which we discuss next.

B. Smolyak Algorithm

One of the central algorithms in the field of multidimensional integration and interpolation is the Smolyak algorithm [25]. Intuitively speaking, the algorithm takes a number of small tensor-product structures and composes them in such a way that the resulting grid has a drastically reduced number of nodes while preserving the approximating power of the full tensor-product construction for the classes of functions that one is typically interested in integrating or interpolating [5].

The Smolyak interpolant for f is as follows:

$$\mathcal{S}_l(f) = \sum_{l-n_\mathrm{d}+1\le|\mathbf{i}|\le l} (-1)^{l-|\mathbf{i}|} \binom{n_\mathrm{d}-1}{l-|\mathbf{i}|} \mathcal{Q}_\mathbf{i}(f) \tag{17}$$

where $l \ge 0$ is the index of the interpolation step, which we shall refer to as the Smolyak level, and $|\mathbf{i}| = i_1 + \cdots + i_{n_\mathrm{d}}$. It can be seen that the algorithm is indeed just a peculiar composition of cherry-picked tensor products. However, the formula has an implication of paramount importance. The function f needs to be evaluated only at the nodes of the grid underpinning (17):

$$\mathcal{Y}_l = \bigcup_{l-n_\mathrm{d}+1\le|\mathbf{i}|\le l} \mathcal{X}_\mathbf{i}. \tag{18}$$

The cardinality of the above set does not have a general closed-form formula; however, it can be several orders of magnitude smaller than the one of the full tensor product given in (16), which depends on the dimensionality of the problem at hand and the one-dimensional rules utilized (Sec. V-A).

A better intuition about the properties of the Smolyak construction can be obtained by rewriting (17) in an incremental form. To this end, let $\Delta\mathcal{Q}_{-1}(f) = 0$,

$$\begin{aligned}\Delta\mathcal{Q}_i(f) &= (\mathcal{Q}_i - \mathcal{Q}_{i-1})(f), \text{ and} \\ \Delta\mathcal{Q}_\mathbf{i}(f) &= \left(\bigotimes_{k=1}^{n_\mathrm{d}} \Delta\mathcal{Q}_{i_k}\right)(f).\end{aligned} \tag{19}$$

Then, (17) is identical to

$$\mathcal{S}_l(f) = \sum_{\mathbf{i}\in\mathcal{I}_l} \Delta\mathcal{Q}_\mathbf{i}(f) = \mathcal{S}_{l-1}(f) + \sum_{\mathbf{i}\in\Delta\mathcal{I}_l} \Delta\mathcal{Q}_\mathbf{i}(f) \tag{20}$$

where $\mathcal{S}_{-1}(f) = 0$, and we let $\mathcal{I}_l = \{\mathbf{i} : |\mathbf{i}| \le l\}$ and $\Delta\mathcal{I}_l = \{\mathbf{i} : |\mathbf{i}| = l\}$. It can be seen that a Smolyak interpolant can be efficiently refined: the work done in order to attain one Smolyak level l is entirely recycled to go to the next.

The sparsity and incremental refinement of the Smolyak approach, which are shown in (18) and (20), respectively, are remarkable properties *per se*, but they can be taken even further. To this end, let $\Delta\mathcal{X}_{-1} = \emptyset$,

$$\Delta\mathcal{X}_i = \mathcal{X}_i \setminus \mathcal{X}_{i-1}, \text{ and } \Delta\mathcal{X}_\mathbf{i} = \Delta\mathcal{X}_{i_1} \times \cdots \times \Delta\mathcal{X}_{i_{n_\mathrm{d}}}.$$

Then, (18) can be rewritten as

$$\mathcal{Y}_l = \bigcup_{\mathbf{i}\in\mathcal{I}_l} \Delta\mathcal{X}_\mathbf{i} = \mathcal{Y}_{l-1} \cup \bigcup_{\mathbf{i}\in\Delta\mathcal{I}_l} \Delta\mathcal{X}_\mathbf{i}, \tag{21}$$

which is analogous to (20). It can be seen now that it is beneficial to the refinement to have $\mathcal{X}_{i-1}$ be entirely included in $\mathcal{X}_i$ since, in that case, the cardinality of $\mathcal{Y}_l \setminus \mathcal{Y}_{l-1} = \bigcup_{\mathbf{i}\in\Delta\mathcal{I}_l} \Delta\mathcal{X}_\mathbf{i}$ derived from (21) decreases. In words, the values of f obtained on lower levels (lower l) can be reused to attain higher levels (higher l) if the grid grows without abandoning its previous structure. With this in mind, the rule used for generating successive sets of points $\{\mathcal{X}_i\}_i$ should be chosen to be nested, that is, in such a way that $\mathcal{X}_i$ contains all nodes of $\mathcal{X}_{i-1}$.

The final step in this subsection is to rewrite (20) in a hierarchical form. To this end, we require the interpolants of higher levels to represent exactly the interpolants of lower levels. In one dimension, it means that

$$\mathcal{Q}_{i-1}(f) = \mathcal{Q}_i(\mathcal{Q}_{i-1}(f)). \tag{22}$$

The condition in (22) can be satisfied by an appropriate choice of collocation nodes and basis functions, which will be discussed later. If (22) holds, using (11) and (19),

$$\Delta\mathcal{Q}_i(f) = \sum_{j\in\Delta\mathcal{J}_i} (f(x_{ij}) - \mathcal{Q}_{i-1}(f)(x_{ij}))\, e_{ij}$$

where $\Delta\mathcal{J}_i = \{j \in \mathcal{J}_i : x_{ij} \in \Delta\mathcal{X}_i\}$. The above sum is over $\Delta\mathcal{X}_i$ due to the fact that the difference in the parentheses is zero whenever $x_{ij} \in \mathcal{X}_{i-1}$ since $\mathcal{X}_{i-1} \subset \mathcal{X}_i$.

In multiple dimensions, using the Smolyak formula,

$$\Delta\mathcal{Q}_\mathbf{i}(f) = \sum_{\mathbf{j}\in\Delta\mathcal{J}_\mathbf{i}} \left(f(\mathbf{x_{ij}}) - \mathcal{S}_{|\mathbf{i}|-1}(f)(\mathbf{x_{ij}})\right) e_\mathbf{ij} \tag{23}$$

where $\Delta\mathcal{J}_\mathbf{i} = \{\mathbf{j} \in \mathcal{J}_\mathbf{i} : \mathbf{x_{ij}} \in \Delta\mathcal{X}_\mathbf{i}\}$. The delta

$$\Delta f(\mathbf{x_{ij}}) = f(\mathbf{x_{ij}}) - \mathcal{S}_{|\mathbf{i}|-1}(f)(\mathbf{x_{ij}}) \tag{24}$$

is called a hierarchical surplus. When increasing the interpolation level, this surplus is nothing but the difference between the actual value of f at a new node and the approximation of this value computed by the interpolant constructed so far.

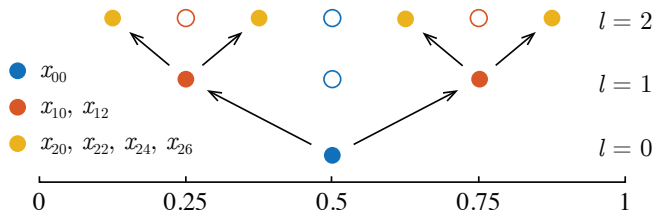

Figure 3. The first three levels of the grid described in Sec. V-C. The empty circles represents the nodes inherited from the levels below.

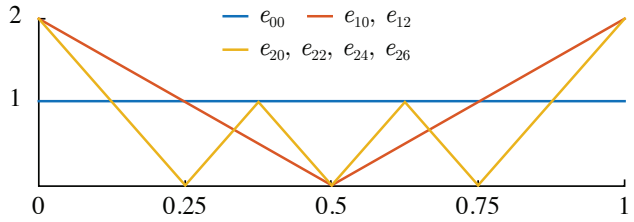

Figure 4. The first three levels of the basis described in Sec. V-D. The basis functions correspond to the collocation nodes show in Fig. 3.

The final formula for nonadaptive hierarchical interpolation is obtained by substituting (23) into (20):

$$\begin{aligned}\mathcal{S}_l(f) &= \sum_{\mathbf{i}\in\mathcal{I}_l}\sum_{\mathbf{j}\in\Delta\mathcal{J}_\mathbf{i}} \Delta f(\mathbf{x_{ij}})\, e_\mathbf{ij} \\ &= \mathcal{S}_{l-1}(f) + \sum_{\mathbf{i}\in\Delta\mathcal{I}_l}\sum_{\mathbf{j}\in\Delta\mathcal{J}_\mathbf{i}} \Delta f(\mathbf{x_{ij}})\, e_\mathbf{ij} \end{aligned} \tag{25}$$

where $\Delta f(\mathbf{x_{ij}})$ is computed according to (24).

C. Collocation Nodes

A sparse grid that is fully nested and, moreover, well disposed to adaptivity, as we shall see, can be constructed using the (one-dimensional) Newton–Cotes rule [6]. For each level, the rule is merely a set of equidistant nodes on $[0, 1]$.

There are two types of the rule: closed and open. The only difference between the two is that the former includes the endpoints, that is, 0 and 1, while the latter does not. Now, in Sec. V-B, we postulated that the assumption in (22) was needed in order to proceed. The closed rule satisfies this assumption, and it is the one used in the original version of local adaptivity presented in [6]. The open Newton–Cotes rule, on the other hand, violates the assumption close to the boundaries of the interval. However, we found that the open rule is a viable option since it performs well in practice, which was also noted in [5]. In fact, we were able to obtain better results with the open rule and decided to present it here.

The open Newton–Cotes rule of level $i \geq 0$ is

$$\mathcal{X}_i = \left\{ x_{ij} = \frac{j+1}{n_i+1} \right\}_{j\in\mathcal{J}_i} \tag{26}$$

where $\mathcal{J}_i = \{i-1\}_{i=1}^{n_i}$ and $n_i = 2^{i+1} - 1$. The first three levels of the rule are depicted in Fig. 3. It can be seen that the number of nodes (in one dimension) grows as 1, 3, 7, 15, 31, and so on, and that the rule is fully nested. In multiple dimensions, the nodes are formed as shown in (13).

D. Basis Functions

The basis functions that correspond the open Newton–Cotes rule described in Sec. V-C are the following piecewise linear functions. For $i = 0$ and $j = 0$, we have that $e_{00}(x) = 1$. 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,

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

The basis functions corresponding to the first three levels of one-dimensional interpolation are depicted in Fig. 4. In multiple dimensions, the basis functions are formed as shown in (14).

Lastly, let us mention the volumes (integrals over the whole domain) of the basis functions denoted by v_{ij}; they will be needed in continuation. Namely, $v_{00} = 1$ and, for $i > 0$,

$$v_{ij} = \int_0^1 e_{ij}(x)\,dx = \begin{cases} \frac{2}{n_i+1}, & \text{if } j \in \{0, n_i - 1\}, \\ \frac{1}{n_i+1}, & \text{otherwise.} \end{cases} \tag{27}$$

In multiple dimensions, the volumes are products of the one-dimensional components, analogous to (15).

Imagine now 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 would require much fewer collocation nodes than the other one; recall Fig. 1. However, if we followed the construction procedure described so far, we would not be able to benefit from this peculiar behavior: we would treat both sides equally and would add all the nodes of each level. The solution to the above problem is to make the interpolation algorithm adaptive, which we shall discuss next.

E. Adaptivity

In order to make the algorithm adaptive, we first need to find a way to measure how good our approximation is at any point in the domain of f. Then, when refining the interpolant, instead of evaluating the function at all possible nodes, we shall only choose those that are located in the regions with poor accuracy as indicated by the yet-to-be-found criterion.

We already have a good foundation for building such a criterion. Recall (24). Hierarchical surpluses are natural indicators of the interpolation error: they are the difference between the values of the true function and those of an approximation at the nodes of the underlying sparse grid. Hence, they can be recycled in order to effectively identify "problematic" regions. Specifically, we first assign a score to each node $\mathbf{x_{ij}}$ or, equivalently, to each pair of level and order indices $(\mathbf{i}, \mathbf{j})$:

$$s_\mathbf{ij} = |\Delta f(\mathbf{x_{ij}})\, v_\mathbf{ij}| \tag{28}$$

where $\Delta f(\mathbf{x_{ij}})$ and $v_\mathbf{ij}$ are given by (24) and (27), respectively, and this score is then used in order to guide the algorithm as we shall explain in the rest of this subsection.

The Smolyak construction in (25) is rewritten as follows:

$$\mathcal{A}_l(f) = \mathcal{A}_{l-1}(f) + \sum_{\mathbf{i}\in\Delta\mathcal{I}_l}\sum_{\mathbf{j}\in\Delta\mathcal{J}_\mathbf{i}} \Delta f(\mathbf{x_{ij}})\, e_\mathbf{ij}. \tag{29}$$

The different with respect to (25) is that $l \geq 0$ is no longer the Smolyak level (see (17)) but a more abstract interpolation step, and $\mathcal{A}_l$ is the interpolant at that step. As always, $\mathcal{A}_{-1} = 0$, and the definition of Δf given in (24) is adjusted accordingly. From now on, all index sets will be generally subsets of their full-fledged counterparts defined in Sec. V-B.

Each $\mathcal{A}_l$ is characterized by a set of level indices $\mathcal{I}_l$, and each $\mathbf{i} \in \mathcal{I}_l$ by a set of order indices $\Delta\mathcal{J}_\mathbf{i}$. At each interpolation step $l \geq 0$, a single index $\mathbf{i}_l$ is chosen from $\mathcal{I}_{l-1}$ with $\mathcal{I}_{-1} = \{\mathbf{0}\}$. The chosen index then gives birth to $\Delta\mathcal{I}_l$ and $\{\Delta\mathcal{J}_\mathbf{i}\}_{\mathbf{i}\in\Delta\mathcal{I}_l}$, which shape the increment in the right-hand side of (29).

The set $\Delta\mathcal{I}_l$ contains the so-called admissible forward neighbors of $\mathbf{i}_l$. Let us now parse the previous sentence. First, the forward neighbors of an index $\mathbf{i}$ are given by

$$\{\mathbf{i} + \mathbf{1}_k : k = 1, \ldots, n_\mathrm{d}\} \tag{30}$$

where $\mathbf{1}_k$ is a vector whose elements are zero except for element k equal to unity. Next, an index $\mathbf{i}$ is admissible if its inclusion into the index set $\mathcal{I}$ in question keeps the set admissible. Finally, $\mathcal{I}$ is admissible if it satisfies the following condition [5]:

$$\mathbf{i} - \mathbf{1}_k \in \mathcal{I}, \text{ for } \mathbf{i} \in \mathcal{I} \text{ and } k = 1, \ldots, n_\mathrm{d}, \tag{31}$$

where, naturally, the cases with $i_k = 0$ need no check.

Now, how is $\mathbf{i}_l$ chosen from $\mathcal{I}_{l-1}$ at each iteration of (29)? First of all, each index can be obviously picked at most once. The rest is resolved by prioritizing the candidates. It is reasonable to assign a priority to a level index $\mathbf{i}$ based on the scores of the order indices associated with it, that is, on the scores of $\mathcal{J}_\mathbf{i}$. We compute the priority as the average score:

$$s_\mathbf{i} = \frac{1}{|\Delta\mathcal{J}_\mathbf{i}|} \sum_{\mathbf{j}\in\Delta\mathcal{J}_\mathbf{i}} s_\mathbf{ij}$$

Consequently, the answer to the above question is that, at each step l, the index $\mathbf{i}$ with the highest $s_\mathbf{i}$ gets promoted to $\mathbf{i}_l$.

Let us now turn to the content of $\Delta\mathcal{J}_\mathbf{i}$ where $\mathbf{i} = \mathbf{i}_l + \mathbf{1}_k$ for a fixed k. It also contains admissible forward neighbors, but they are order indices, and their construction is drastically different from the one in (30). Concretely, these indices are identified by inspecting the backward neighborhood of $\mathbf{i}$ (analogous to (30)). For each backward neighbor $\hat{\mathbf{i}} = \mathbf{i} - \mathbf{1}_{\hat{k}}$ and each $\mathbf{j} \in \Delta\mathcal{J}_{\hat{\mathbf{i}}}$, we begin by checking the following condition: $s_{\hat{\mathbf{i}}\mathbf{j}} \geq \epsilon_s$ where ϵ_s is a user-defined constant, which we shall refer to as the score error. If the condition holds, the forward neighbors of $\mathbf{j}$ in dimension k are added to $\Delta\mathcal{J}_\mathbf{i}$. This procedure is illustrated in Fig. 3 for the open Newton–Cotes rule (see Sec. V-C). The arrows emerging from a node connect the node with its forward neighbors. It can be seen that each node has two forward neighbors (for each dimension); their order indices are

$$(j_1, \ldots, 2j_k, \ldots, j_{n_\mathrm{d}}) \quad \text{and} \quad (j_1, \ldots, 2j_k + 2, \ldots, j_{n_\mathrm{d}}).$$

The above refinement procedure is repeated for each index $\mathbf{i} \in \Delta\mathcal{I}_l$ with respect to each dimension $k = 1, \ldots, n_\mathrm{d}$.

The final question is the stopping condition of the approximation process in (29). Apart from the natural constraints on the maximum number of function evaluations and the maximum allowed Smolyak level l in (17), we rely on the following criterion. Assume that we are given two additional constants: ϵ_a and ϵ_r referred to as the absolute and relative error, respectively. Then, the process is terminated as soon as

$$\max_{(\mathbf{i},\mathbf{j})} |\Delta f(\mathbf{x}_\mathbf{ij})| \leq \max\{\epsilon_a,\, \epsilon_r(f_\mathrm{max} - f_\mathrm{min})\} \tag{32}$$

where f_min and f_max are the minimum and maximum observed value of f, respectively, and the left-hand side is the maximum surplus whose level index has not been refined yet (considered as $\mathbf{i}_l$ at some step l in (29)). The above criterion is a sound way to curtail the process as it is based on the actual progress.

The adaptivity presented in this subsection is referred to as hybrid as it combines features of global and local adaptivity; the combination was proposed in [4]. Local adaptivity, which has already been sufficiently motivated, is due to [6], and it operates on the level of individual nodes. Global adaptivity is due to [5], and it operates on the level of individual dimensions. The intuition behind global adaptivity is that, in general, the input variables manifest themselves (impact f) differently, and the interpolation algorithm is likely to benefit by prioritizing those variables that are the most influential.

To summarize, we have obtained an efficient algorithm for adaptive hierarchical interpolation in multiple dimensions. The main equation is (29) where Δf, $\mathbf{x}_\mathbf{ij}$, and $e_\mathbf{ij}$ are the ones given in Sec. V-B, Sec. V-C, and Sec. V-D, respectively, and the interpolation procedure is undertaken according to the rules given in Sec. V-E. In the next subsection, we shall discuss the algorithmic structure of our interpolation.

F. Implementation

The life cycle of interpolation has roughly two stages: construction and usage. The construction stage invokes f at a set of collocation nodes and produces certain artifacts. The usage stage estimates the values of f at a set of arbitrary points by manipulating the artifacts. In this subsection, we shall look at the pseudocodes of the two stages. The purpose is to give the big picture. All the details can be found online [18].

Let us first make a general note. We found it beneficial to the clarity and ease of implementation to collapse the two sums in (29) into one. This requires storing a level index $\mathbf{i} = (i_k)_{k=1}^{n_\mathrm{d}}$ and an order index $\mathbf{j} = (j_k)_{k=1}^{n_\mathrm{d}}$ for each interpolation element. 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 single vector $\boldsymbol{\iota} = (\iota_k)_{k=1}^{n_\mathrm{d}}$, which we simply call an index. The encoding that we utilize is as follows: $\iota_k = i_k \vee (j_k \ll n_\mathrm{bits})$ where $\vee$ and $\ll$ are the bitwise OR and logical left shift, respectively, and n_bits is the number of bits reserved for storing Smolyak levels (see (17)), which can be adjusted according to the maximum permitted deepness of interpolation.

The pseudocode of the construction stage is given in Algorithm 1 called `Construct`. The `target` input is a function f to be approximated. The `surrogate` output is a structure containing the artifacts of interpolation, which are a set of tuples $\{(\boldsymbol{\iota}_k, \Delta f(\mathbf{x}_{\boldsymbol{\iota}_k})\}_k$, giving a comprehensive description of an interpolant. The routine works as follows.

Line 2: Each iteration is an interpolation step in (29). It has a state captured by a structure denoted by `s`. The `strategy` object represents an adaptation strategy utilized and works as described in Sec. V-E. The `First` method of `strategy` returns the initial state of the first step so that the `indices` field of `s`

Algorithm 1 *Construct* an interpolant of a function.

Input: target // Function to interpolate
Output: surrogate // Interpolant

```
1:  surrogate = New()
2:  for s = strategy.First(); s != nil; s = strategy.Next(s) do
3:      s.nodes = grid.Compute(s.indices)
4:      s.values = Invoke(target, s.nodes)
5:      s.estimates = Evaluate(surrogate, s.nodes)
6:      s.surpluses = Subtract(s.values, s.estimates)
7:      s.scores = strategy.Score(s.surpluses)
8:      Append(surrogate, s.indices, s.surpluses)
9:  end for
10: return surrogate
```

Algorithm 2 *Evaluate* an interpolant at a set of points.

Input: surrogate, points // Interpolant and points
Output: estimates // Approximated values

```
1:  estimates = New()
2:  for point in points do
3:      estimate = 0
4:      for (index, surplus) in surrogate do
5:          weight = basis.Compute(index, point)
6:          estimate = estimate + surplus × weight
7:      end for
8:      Append(estimates, estimate)
9:  end for
10: return estimates
```

is initialized with the indices of that step. The body of the loop populates the rest of the fields of `s` so that `strategy.Next` can adequately produce the initial state of the next iteration. The process terminates when a stopping condition is satisfied, in which case `Next` returns a null state.

Line 3: The `grid` object represents the interpolation grid utilized (see Sec. V-C), and its `Compute` method converts the step's indices into the coordinates of the corresponding collocation nodes, that is, $\{\boldsymbol{\iota}_k\}_k$ into $\{\mathbf{x}_{\boldsymbol{\iota}_k}\}_k$.

Line 4: `Invoke` evaluates `target` at the collocation nodes. This is by far the most time consuming function of the algorithm as `target` is generally expensive to evaluate. This function is also a prominent candidate for parallelization since the algorithm does not impose any evaluation order.

Line 5: `Evaluate` exercises the interpolant constructed so far at the collocation nodes, approximating the values obtained on line 4. This function will be discussed separately.

Line 6: `Subtract` computes the difference between the true and approximated values of `target`, which yields the step's hierarchical surpluses $\{\Delta f(\mathbf{x}_{\boldsymbol{\iota}_k})\}_k$, similar to (24).

Line 7: `strategy.Score` calculates the scores of the new collocation nodes based on their surpluses; see (28).

Line 8: `Append` improves the interpolant by extending it with the indices and surpluses of the current iteration.

We now turn to the usage stage of an interpolant. The pseudocode is given in Algorithm 2 called `Evaluate`. This algorithm is also involved in Algorithm 1; see line 5. Let us make a couple of observations regarding `Evaluate`.

Line 4: The inner loop is an unfolded version of (29) (there is no separation between individual interpolation steps taken).

Line 5: The `basis` object represents the interpolation basis utilized (see Sec. V-D), and its `Compute` method evaluates a single (multidimensional) basis function at a single point.

It is worth noting that the `basis`, `grid`, and `strategy` objects conform to certain interfaces and can be easily swapped out. This makes the two algorithms 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, we have presented the key component of the proposed framework for probabilistic analysis of electronic systems: an efficient approach to multidimensional interpolation. The overall technique has been consolidated in Algorithm 1 and Algorithm 2.

VI. Analysis

In Sec. IV, we formalized the uncertainty affecting electronic systems and discussed several aspects of such systems along with metrics g, which the designer is interested in evaluating. In Sec. V, we obtained an efficient interpolation algorithm for approximating hypothetical multidimensional functions f. We shall now amalgamate the ideas developed in the aforementioned two sections.

Given an electronic system dependent on a number of uncertain parameters $\mathbf{u} : \Omega \to \mathbb{R}^{n_u}$, the goal is to analyze a metric g representing a certain aspect of the system. For instance, $\mathbf{u}$ can correspond to the execution times of the tasks, and g can correspond the total energy consumed by the processing elements, as we exemplify in Sec. IV-B and Sec. IV-C. The goal is attained as follows; recall Fig. 2.

1) The parametrization of g is changed from $\mathbf{u}$ to random variables $\mathbf{z} : \Omega \to [0,1]^{n_z}$ via a suitable transformation $\mathbb{T}$; this stage is described in Sec. IV-A. 2) An interpolant of the resulting composition $g \circ \mathbb{T}$ is constructed by treating the composition as a deterministic function f of $\mathbf{z}$; this stage is detailed in Sec. V. 3) An estimation of the probability distribution of g is undertaken in the usual sampling-based manner but relying solely on the constructed interpolant; g is no longer involved. This last stage boils down to drawing independent samples from $F_{\mathbf{z}}$ and evaluating the interpolant $\mathcal{A}_l(f) \equiv \mathcal{A}_l(g \circ \mathbb{T})$ at those points. Having collected samples of g, other statistics about g, such as probabilities of particular events, can be straightforwardly estimated. We do not discuss this estimation stage any further as it is standard.

There are two aspects concerning the usage of the proposed framework that we would like to cover in what follows.

A. Expectation and Variance

Since the expected value and variance, which are defined in (1) and (2), respectively, usually draw particular attention, we would like to elaborate on them separately.

As shown in Sec. IV-A, g can be reparameterized in terms of independent variables that are uniformly distributed on $[0,1]^{n_z}$. This means that the probability density function of $\mathbf{z}$ simply equals to one. Therefore, using (1) and (29), we have

$$\mathbb{E}\, g \approx \mathbb{E}\, \mathcal{A}_l(f) = \int_{[0,1]^{n_z}} \mathcal{A}_l(f)(\mathbf{z})\, d\mathbf{z} = \sum_{\mathbf{i}\in\mathcal{I}_l} \sum_{\mathbf{j}\in\Delta\mathcal{J}_\mathbf{i}} \Delta f(\mathbf{x_{ij}})\, v_\mathbf{ij}$$

where

$$v_\mathbf{ij} = \int_{[0,1]^{n_z}} e_\mathbf{ij}(\mathbf{z})\, d\mathbf{z} = \prod_{k=1}^{n_z} \int_0^1 e_{i_k j_k}(z_k)\, dz_k = \prod_{k=1}^{n_z} v_{i_k j_k}.$$

In the above equation, v_{ij} is as shown in (27). Consequently, we have obtained an analytical formula for the expected value of g, which does not require any additional sampling.

Regarding the variance of g, it can be seen in (2) that the variance can be assembled from two components: the expected value of g, which we already have, and the expected value of g^2, which we are missing. The solution is to let $h = (g, g^2)$ be the metric instead of g. Then the expected values of both g and g^2 will be available in analytical forms, and the variance of g can be computed using (2). This approach can be generalized to probabilistic moments of higher orders.

B. Multiple Outputs

The careful reader has noted a problem with the calculation of variance in the previous subsection: h is vector valued. More generally, the metric g in Sec. IV and the function f in Sec. V have been depicted as having one-dimensional codomains. This, however, has been done only for the sake of clarity. All the mathematics and pseudocodes stay the same for vector-valued functions. The only except is that, since a surplus $\Delta f(\mathbf{x_{ij}})$ naturally inherits the output dimensionality of f, the operations that involve $\Delta f(\mathbf{x_{ij}})$ should be adequately adjusted. If the outputs are on different scales and/or have different accuracy requirements, one might want to have different ϵ_a and ϵ_r in (32) for different outputs. In that case, one also needs to device a more sensible strategy for scoring collocation nodes in (28) such as rescaling individual outputs and then calculating the uniform norm $\|\cdot\|_\infty$ or L^2 norm $\|\cdot\|_2$. Our code [18] has been written with multiple outputs in mind.

To summarize, once an interpolant of g has been constructed, the distribution of g is estimated using versatile sampling methods applied to the interpolant. The framework extends naturally to metrics with multiple outputs, and it provides analytical formulae for expectations and variances.

Let us remind that the evaluation of g is an extensive operation. Our technique is designed to keep this expense as low as possible by choosing the evaluation points adaptively, which is unlike traditional sampling methods. Moreover, in contrast to PC expansions and similar techniques, the proposed framework is well suited for the nonsmooth response surfaces.

VII. Experiments

In this section, we evaluate the performance of our framework. Our implementation is open source and can be found at [18], which also includes the experimental setup along with configuration files and input data. The experiments discussed below are conducted on a GNU/Linux machine equipped with 16 processors Intel Xeon E5520 2.27 GHz and 24 GB of RAM.

We shall address $3 \times 2 \times 3 = 18$ uncertainty-quantification problems. Specifically, we shall consider three platform sizes n_π: 2, 4, and 8 processing elements; two application sizes n_t: 10 and 20 tasks; and three metrics g: the end-to-end delay, total energy consumption, and maximum temperature defined in (6), (8), and (10), respectively. At this point, it might be helpful to recall the example in Fig. 2.

A. Configuration

A platform with n_π processing elements and an application with n_t tasks are generated randomly by the TGFF tool [26]. The tool generates n_π tables and a directed acyclic graph with n_t nodes. Each table corresponds to a processing element, and it describes certain properties of the tasks when they are mapped to that particular processing element. Namely, each table assigns two numbers to each task: a reference execution time, chosen uniformly between 10 and 50 ms, and a power consumption, chosen uniformly between 5 and 25 W. The graph captures data dependencies between the tasks. The application is scheduled using a list scheduler [27]. 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).

The construction of thermal RC circuits needed for temperature analysis is delegated to the HotSpot tool [24]. The floorplan of each platform is a regular grid wherein each processing element occupies 2×2 mm^2 on the die. The output of the tool is a pair of a thermal capacitance matrix $\mathbf{C}$ and a thermal conductance $\mathbf{G}$ matrix used in (9). The leakage modeling is based on a linear fit to a data set of SPICE simulations of a series of CMOS invertors [17], [23]; see also [11]. The time step of power and temperature profiles is constant and equal to one microsecond; see Sec. IV-C and Sec. IV-D.

The uncertain parameters $\mathbf{u}$ introduced in Sec. III-B are the execution times of the tasks; see Sec. IV-B. All other parameters are deterministic. Targeting the practical scenario described in Sec. IV-A, the marginal distributions and correlation matrix of $\mathbf{u}$ are assumed to be available. Without loss of generality, the marginal of u_i is a four-parametric beta distribution $\mathrm{Beta}(\alpha_i, \beta_i, a_i, b_i)$ where α_i and β_i are the shape parameters, and a_i and b_i are the endpoints of the support. The left a_i and right b_i endpoint are set to 80% and 120%, respectively, of the reference execution time generated by the TGFF tool as described earlier. The parameter α_i and β_i are set to two and five, respectively, for all tasks, which skews the distribution toward the left endpoint. The execution times of the tasks are correlated based on the structure of the graph produced by the TGFF tool: 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 stronger u_i and u_j are correlated. The model-order reduction parameter η in (4) (Sec. IV-A) is set to 0.9, which results in $n_z = 2$ and 3 preserved variables for applications with $n_t = 10$ and 20 tasks, respectively.

The configuration of the interpolation algorithm (the collocation nodes, basis functions, and adaptation strategy with stopping conditions) is as described in Sec. V. The parameters ϵ_a, ϵ_r, and ϵ_s are around 10^3, 10^2, and 10^4, respectively, depending on the problem; the exact values can be found at [18], which, again, contains all other details too.

The performance of our framework with respect to each problem is assessed as follows. First, we obtain the "true" probability distribution of the metric in question g by sampling

g directly and extensively. Direct sampling means that samples are drawn from g itself (not from a surrogate), and that there is no any intermediate model-order reduction (see Sec. IV-A). Second, we construct an interpolant for g and estimate g's distribution by sampling the interpolant. In both cases, we draw 10^5 samples; let us remind, however, that the cost of sampling the interpolant is practically negligible. Third, we perform another round of direct sampling of g, but this time we draw as many samples as many times the metric was evaluated during the interpolation process. In each of the three cases, the sampling is undertaken in accordance with a Sobol sequence, which is a quasi-random low-discrepancy sequence featuring much better convergence properties than those of the classical Monte-Carlo (MC) sampling [28].

As a result, we obtain three estimates of g's distribution: reference (the one considered true), proposed (the one interpolation powered), and direct (the one equal in terms of the number of g's evaluations to the proposed solution). The last two are compared with the first one. For comparing the proximity between two distributions, we use the well-known Kolmogorov–Smirnov (KS) statistic [29], which is the supremum over the distance (pointwise) between two empirical distribution functions and, hence, is a rather unforgiving error indicator.

B. Discussion

The results of all 18 uncertainty-quantification problems are given in Fig. 5 as a 6-by-3 grid of plots, one plot per problem. The three columns correspond to the three metrics at hand: the end-to-end delay (left), total energy (middle), and maximum temperature (right). The three pairs of rows correspond to the three platform sizes: 2 (top), 4 (middle), and 8 (bottom) processing elements. The rows alternate between the two application sizes: 10 (odd) and 20 (even) tasks.

The horizontal axis of each plot shows the number of points, that is, evaluations of the metric g, and the vertical one shows the KS statistic on a logarithmic scale. Each plot has two lines. The solid line represents our technique. The circles on this line correspond to the steps of the interpolation process given in (29). They show how the KS statistic computed with respect to the reference solution changes as the interpolation process takes steps (and increases the number of collocation nodes) until the stopping condition is satisfied (Sec. V-E). Note that only a subset of the actual steps is displayed in order to make the figure legible. Synchronously with the solid line (that is, for the same numbers of g's evaluations), the dashed line shows the error of direct sampling, which, as before, is computed with respect to the reference solution.

Let us first describe one particular problem shown in Fig. 5. Consider, for instance, the one labeled with ★. It can be seen that, at the very beginning, our solution and the solution of direct sampling are poor. The KS statistic tells us that there are substantial mismatches between the estimates and the reference solution. However, as the interpolant is being adaptively refined, our solution approaches rapidly the reference one and, by the end of the interpolation process, leaves the solution of naïve sampling approximately an order of magnitude behind.

Studying Fig. 5, one can make a number of observations. First and foremost, our interpolation-powered approach (solid lines) to probabilistic analysis outperforms direct sampling (dashed lines) in all the cases. This means that, given a fixed budget of the computation time—the probability distributions delivered by our framework are much 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 direct sampling methods try to cover the probability space impartially, Fig. 5 is a salient illustration of the difference between being adaptive and nonadaptive.

It can also be seen in Fig. 5 that, as the number of evaluations increases, the solutions computed by our technique approach the exact ones. The error of our framework decreases generally steeper than the one of direct sampling. The decrease, however, tends to plateau toward the end of the interpolation process (when the stopping condition is satisfied). This behavior can be explained by the following two reasons. First, the algorithm has been instructed to satiate certain accuracy requirements (ϵ_a, ϵ_r, and ϵ_s), and it reasonably does not do more than what has been requested. Second, since the model-order reduction mechanism is enabled in the case of interpolation, the metric being interpolated is not g, strictly speaking; it is a lower-dimensional representation of g, which already implies an information loss. Therefore, there is a 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 the classical MC sampling, the switch might lead to even more dramatic savings than those shown in Fig. 5. Needless to mention that the gain is especially prominent in situations where the analysis needs to be performed many times such as when it resides in a design-space exploration loop.

Remark 3. *The wall-clock time taken by the experiments is not reported in this paper because this time is irrelevant: since the evaluation of g is time consuming (see Sec. III-B), the number of g's evaluations is the most apposite expense indicator. For the curious reader, however, let us give an example by considering the problem labeled with ♣ in Fig. 5. Obtaining a reference solution with 10^5 simulations in parallel on 16 processors took us around two hours. Constructing an interpolant with 383 collocation nodes took around 30 seconds (this is also the time of direct sampling with 383 simulations of g). Evaluating the interpolant 10^5 times took 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 proportional to g's evaluation time.*

C. Real-life Example

Last but not least, we investigate the viability of deploying the proposed framework in a real environment. It means that we need to couple the framework with a battle-proven simulator, which is used in both academia and industry, and let it simulate a real application running on a real platform. Before we proceed, we would like to remind that all the implementation and configuration details including the infrastructure developed for this example can be found at [18].

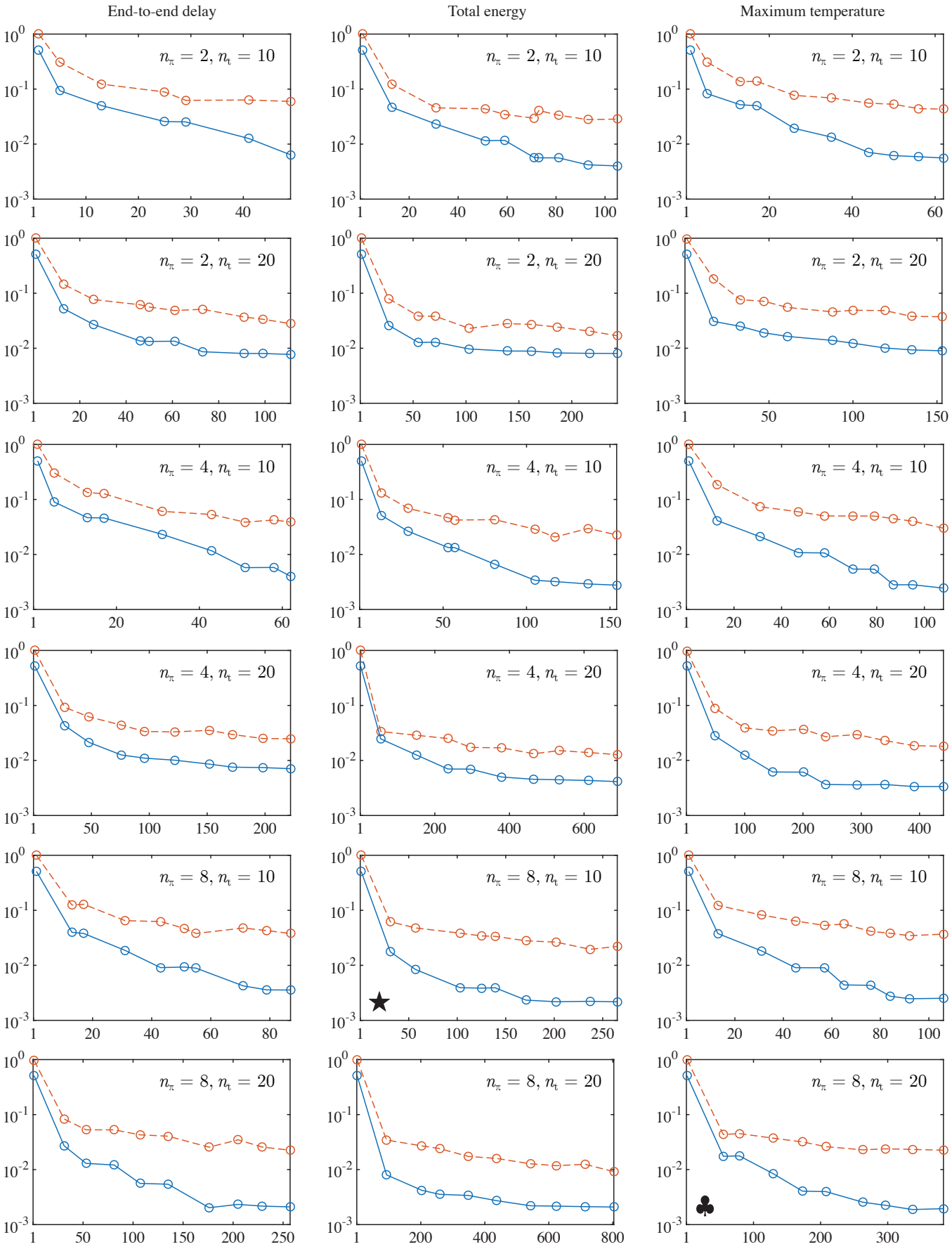

Figure 5. Experimental results. The columns correspond to the metrics written at the very top. The horizontal axes show the number of points, and the vertical ones the error on logarithmic scales. The solid lines correspond to the proposed technique, and the dashed ones to direct sampling.

The scenario that we consider is the same as the one depicted in Fig. 2 except for the fact that an industrial-standard simulator is put in place of the "black box" on the left side, and that the metric of interest g is now the total energy. Unlike the previous examples, there is no true solution to compare with due to the prohibitive expense of the simulator, which is exactly why our framework is needed in such cases.

The simulator of choice is the well-known and widely used combination of Sniper [21] and McPAT [30]. The architecture that we simulate is Intel's Nehalem-based Gainestown series. Sniper is distributed with a configuration file for this architecture, and we use it without any changes. The platform is configured to have three CPUs sharing one L3 cache.

The application that has been chosen for simulation is VIPS, which is an image-processing piece of software taken from the PARSEC benchmark suite [31]. 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 $\mathbf{u}$ (see Sec. III-B), which are assumed to be distributed uniformly within certain ranges.

The real-life deployment has fulfilled our expectations. The interpolation process successfully finished and delivered a surrogate after 78 invocations of the simulator. Each invocation took 40 minutes on average. The probability distribution of the total energy was then estimated by sampling the constructed surrogate 10^5 times. These many samples would take around 6 months to obtain on our machine if we sampled the simulator directly in parallel on 16 processors; using the proposed technique, the whole procedure took approximately 9 hours.

VIII. Conclusion

In this paper, we have presented a framework for probabilistic analysis of electronic systems. Given a description of the probability distribution of the uncertain parameters present in the system under consideration and a simulator of a metric of interest dependent on the parameters, the framework prescribes the steps that need to be taken in order to computationally efficiently obtain the probability distribution of the metric.

The proposed approach is powered by hierarchical interpolation following a hybrid adaptation strategy. The adaptivity makes the framework particularly suited for problems with idiosyncratic behaviors and steep response surfaces, which arise in electronic systems due to their digital nature.

The performance of our framework has been assessed by comparing it with the performance of an advanced sampling technique. The experimental results have shown that, for a fixed budget of evaluations of the metric, our approach achieves higher accuracy compared to direct simulations.

Finally, we would like to emphasize that, even though the framework has been exemplified by considering a specific source of uncertainty and specific metrics, it is general and can be successfully applied in many other settings.

Acknowledgments

The authors would like to thank Prof. Nicholas Zabaras and Dr. Xiang Ma from Cornell University and Dr. Stefan Dirnstorfer from Munich Technical University for the help with adaptive hierarchical interpolation and integration techniques.

References

[1] S. Quinton, R. Ernst, D. Bertrand, and P. M. Yomsi, "Challenges and new trends in probabilistic timing analysis," in *DATE*, 2012, pp. 810–815.
[2] S. Asmussen and P. Glynn, *Stochastic Simulation: Algorithms and Analysis*. Springer New York, 2007.
[3] I. Díaz-Emparanza, "Is a small Monte Carlo analysis a good analysis?" *Statistical Papers*, vol. 43, pp. 567–577, October 2002.
[4] J. Jakeman and S. Roberts, "Local and dimension adaptive stochastic collocation for uncertainty quantification," in *Sparse Grids and Applications*. Springer Berlin Heidelberg, 2012, pp. 181–203.
[5] W. A. Klimke, "Uncertainty modeling using fuzzy arithmetic and sparse grids," Ph.D. dissertation, Universität Stuttgart, 2006.
[6] X. Ma and N. Zabaras, "An adaptive hierarchical sparse grid collocation algorithm for the solution of stochastic differential equations," *J. Computational Physics*, vol. 228, pp. 3084–3113, May 2009.
[7] A. Srivastava, D. Sylvester, and D. Blaauw, *Statistical Analysis and Optimization for VLSI: Timing and Power*. Springer US, 2005.
[8] 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," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, pp. 1812–1825, October 2008.
[9] D.-C. Juan, Y.-L. Chuang, D. Marculescu, and Y.-W. Chang, "Statistical thermal modeling and optimization considering leakage power variations," in *DATE*, 2012, pp. 605–610.
[10] Y.-M. Lee and P.-Y. Huang, "An efficient method for analyzing on-chip thermal reliability considering process variations," *ACM Trans. Design Automation of Electronic Systems*, vol. 18, pp. 41:1–41:32, July 2013.
[11] I. Ukhov, P. Eles, and Z. Peng, "Probabilistic analysis of power and temperature under process variation for electronic system design," *IEEE Trans. CAD Integr. Circuits Syst.*, vol. 33, pp. 931–944, June 2014.
[12] ——, "Temperature-centric reliability analysis and optimization of electronic systems under process variation," *IEEE Trans. VLSI Syst.*, vol. 23, pp. 2417–2430, November 2015.
[13] J. L. Díaz, D. F. García, K. Kim, C.-G. Lee, L. L. Bello, J. M. López, S. L. Min, and O. Mirabella, "Stochastic analysis of periodic real-time systems," in *RTSS*, 2002, pp. 289–300.
[14] L. Santinelli, P. M. Yomsi, D. Maxim, and L. Cucu-Grosjean, "A component-based framework for modeling and analyzing probabilistic real-time systems," in *ETFA*, 2011, pp. 1–8.
[15] B. Tanasa, U. Bordoloi, P. Eles, and Z. Peng, "Probabilistic response time and joint analysis of periodic tasks," in *ECRTS*, 2015, pp. 235–246.
[16] D. Xiu, *Numerical Methods for Stochastic Computations: A Spectral Method Approach*. Princeton University Press, 2010.
[17] I. Ukhov, M. Bao, P. Eles, and Z. Peng, "Steady-state dynamic temperature analysis and reliability optimization for embedded multiprocessor systems," in *DAC*, 2012, pp. 197–204.
[18] (2017, February) Source code and input data used in the experimental results. Embedded Systems Laboratory at Linköping University. [Online]. Available: https://www.ida.liu.se/~ivauk83/research/PAAI
[19] R. Durrett, *Probability*. Cambridge University Press, 2010.
[20] R. Nelsen, *An Introduction to Copulas*. Springer, 2007.
[21] 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*, November 2011, pp. 52:1–52:12.
[22] P.-L. Liu and A. D. Kiureghian, "Multivariate distribution models with prescribed marginals and covariances," *Probabilistic Engineering Mechanics*, vol. 1, pp. 105–112, June 1986.
[23] Y. Liu, R. P. Dick, L. Shang, and H. Yang, "Accurate temperature-dependent integrated circuit leakage power estimation is easy," in *DATE*, 2007, pp. 1526–1531.
[24] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *ACM Trans. Architecture and Code Optimization*, vol. 1, pp. 94–125, March 2004.
[25] S. A. Smolyak, "Quadrature and interpolation formulas for tensor products of certain classes of functions," *Doklady Akademii Nauk SSSR*, vol. 148, pp. 1042–1045, 1963.
[26] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: Task graphs for free," in *CODES/CASHE*, March 1998, pp. 97–101.
[27] T. L. Adam, K. M. Chandy, and J. R. Dickson, "A comparison of list schedules for parallel processing systems," *Communications of the ACM*, vol. 17, pp. 685–690, December 1974.
[28] S. Joe and F. Kuo, "Constructing sobol sequences with better two-dimensional projections," *SIAM Journal on Scientific Computing*, vol. 30, pp. 2635–2654, 2008.
[29] C. R. Rao, *Linear Statistical Inference and Its Applications*. John Wiley & Sons, 2009.
[30] 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*, December 2009, pp. 469–480.
[31] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, January 2011.