Building Faithful Embedded Systems Models: Challenges and Opportunities

Ayoub Nouri[1], Marius Bozga[2], and Saddek Bensalem[2]

[1] Univ. Grenoble Alpes, F-38000 Grenoble, France
CEA, LETI, MINATEC Campus, F-38054 Grenoble, France
[2] Université Grenoble Alpes, VERIMAG, F-38000 Grenoble, France
CNRS, VERIMAG, F-38000 Grenoble, France

Abstract. In this chapter, we overview some of the sought challenges for building faithful embedded systems models. We highlight the growing demand for using formal models especially for dealing with performance. The chapter illustrates the impact of the hardware part of the system on performance and suggests a probabilistic interpretation in order to build appropriately abstract models towards trustworthy analysis. We believe that such a view is worth to investigate to faithfully characterize the system performance as it provides a formal and parsimonious framework. In this context, we survey some probabilistic models and techniques that we think interesting for building such faithful representations.

1 Introduction

Embedded systems (ES) have deeply impacted our daily lives and contributed to draw a completely new lifestyle where mobility, rapidity, and connectivity are the keywords. The substantial advances in the integrated circuits and the networks bandwidth have contributed to democratize these systems which became ubiquitous. According to the Artemis Strategic Research Agenda 2011[3], it is estimated that there will be over 40 billion devices worldwide, that is 5 to 10 embedded devices per person on earth by 2020. While a large portion of ES is dedicated for customer electronics (entertainment and personal use), they are becoming also essential for companies and even for governments and states as they represent an important leverage for innovation and competitiveness. Domains benefiting from ES assets, such as transportation, national security, health care, and education, to mention but a few, are wide and steadily increasing.

The great and various benefits of ES come at the price of an increasing complexity to design them. More burden is thus put on designers that have to produce systems in less time with ever reducing costs. Whereas designing mixed hardware/software systems that provide sophisticated services is inherently challenging, additional constraints such as the shrinking time to market and costs optimization make it even harder. Furthermore, ES are increasingly

[3] ftp.cordis.europa.eu/pub/technology-platforms/docs/sra-2011-book-page-by-page-9.pdf

used in safety-critical setups involving human lives and wealth. Thus, formally ensuring their functional correctness is becoming primordial.

As our reliance on these systems increases, so does the expectation that they will provide us with more mobility and ensure high-performance response. The major breakthrough of ES in our everyday lives, has resulted on a shift from task-specific settings, e.g. control of an industrial robot, to more programmable configurations running various functionalities, targeting mass consumption, and often battery supplied. The former rely on dedicated hardware ensuring low power consumption, low cost, real-time constraints, and silicon efficiency, whereas the latter require programmable circuits with an increasing computation power, which are less efficient, consume more energy, and are harder to program.

The design of these systems generally falls within one of three configurations[4]: (i) *hardware-centric*: which aims at finding the most appropriate hardware architecture for a specific domain of application, e.g., multimedia, (ii) *software-centric*: which tackles the issue of designing and deploying functionally correct and efficient applications on a given hardware architecture, and (iii) *co-design*: which starts from abstract functional specifications and tries to figure out the best partition of these functionalities into software and hardware components.

In this chapter, we focus on the design of embedded software applications for a given hardware architecture [31]. Whereas several advances have been performed in this context [29, 35, 57, 50, 25, 28], many open challenges are still ahead, especially with the advent of multi- and many-cores architectures, e.g. how to distribute and map software applications onto these platforms for an optimal utilization and to satisfy performance requirements, e.g. energy?

This chapter considers the design of embedded software following a model-based approach and using formal techniques. In model-based design, the whole process is driven by models and all the choices are made upon them. Hence, they must satisfy two important conditions for a successful design. First, they must be faithful, that is to capture the real behavior of the system, in term of functionality and performance. Second, the built models must have a clear interpretation, i.e. they must have a formally defined semantics, to enable unambiguous/trustworthy analysis and then enable to make well founded choices. Relying on models that do not respect these conditions leads to inconsistent decisions and later to poor designs.

In this work, our concern are the earliest phases of the design as they have the greatest impact on the rest of the process. We precisely focus on modeling performance aspects, which are inherently challenging and hard to faithfully predict during the early stages of ES design. Unlike classical programs design where performance is evaluated using complexity theory with respect to abstract machine, i.e. Turing machine, the performance of ES is induced by the combination of the software and hardware parts of the system, e.g. the execution time of a Fourier transform on a processing unit, the communication delay of a bus or a Network on Chip (NoC), the amount of consumed energy induced by that function on the corresponding hardware.

[4] These settings may coexist withing the same design process, e.g. at different phases.

Building faithful, formal, and high-level system models that capture the functional and the performance aspects entails several challenges, e.g. how to access (predict) the real performance evolution of the system during the early design phases? Assuming that such information is available, what is the appropriate model to characterizes it faithfully, and how? Additional challenges concern the appropriate level of abstraction of the performance model. In addition of being determinant for applying formal analysis techniques, the choice of the adequate level of abstraction is essential to enable understanding and mastering the complexity of the system under design, especially early in the process.

An important characteristic that must be considered when dealing with performance of ES is variability. ES performance is ideally constant, albeit in practice, it shows fluctuation, which should be taken into consideration for a trustworthy analysis. The environment where the ES is deployed has an important impact on performance and it can be seen twofold. First, is the input data to the system, e.g. different video qualities for a multimedia player. Variable inputs generally provokes a variable performance, although some systems expose a constant behavior when varying input data (data-independent systems). The second component within the environment includes all the external factors which may affect its function and/or performance, e.g. unusual external temperature. Another important source of fluctuation, which will be our focus in this chapter, is the hardware (and networked) part of the system, e.g. buses and memory.

Ideally, the variability induced by the above factors is deterministically characterized, which implies to build detailed models of the environment and the hardware architecture. However, besides being in contradiction with the objective of building high-level models, this is generally unfeasible. Precisely modeling the environment (including inputs) is unfeasible unless the system is designed for specific tasks and is targeted to a well known and controlled environment. Similarly, building formal detailed models of the underlying hardware is challenging. First, it requires to have detailed specifications, which are rarely available during early design phases. Assuming that such details are available, this will induce an important understanding/interpretation effort and consequently a considerable time and produces huge models. This eventually leads to an ad hoc approach which is tedious and error prone. Abstraction of details is thus a must. That is, we need models and methods that enable characterizing performance[5] and to use it for building formal high-level system models (including the functional behavior) towards trustworthy analysis.

The vision supported in this work is to interpret the performance evolution probabilistically. We argue that such a view provides a natural and faithful abstraction in addition of being the unique mathematically sound framework for capturing variability and its uncertainty. Probabilistic modeling is a natural choice to abstract details either because we don't care about them for the moment, e.g. early design stages, or because we are not able to handle all of them, e.g. no access, too complex, or it takes too much time.

[5] Our view of characterizing performance is to capture the gist of performance evolution formally and in a parsimonious way

The chapter aims at providing, without claiming to be exhaustive, an overview of the sought challenges and promising techniques for building faithful, formal, high-level probabilistic software/hardware models for performance evaluation. This work is motivated by a steadily growing research results providing efficient formal probabilistic techniques that can be used for performance evaluation [63, 27, 5]. We consider that among the obstacle for using such techniques are the absence of systematic techniques to obtain faithful formal models. We detail in Section 2 the issues related to faithfully characterizing ES performance. Then we review in Section 3 some existing methods and models for gathering and characterizing performance. In Section 4, we overview a recent work that provides an attempt to answer the underlying challenges following a probabilistic view. We discuss the assumptions and restrictions made in this work and provide in Section 5 future directions and sought opportunities to overcome them.

2 Challenges for ES Performance

Performance of ES can be thought in term of several aspects, e.g. timing behavior (computation and communication), energy consumption, memory utilization, or global throughput. These aspects are necessary together with functional behavior to enable trustworthy system analysis. In this section we highlight some factors that may affect ES performance. We precisely focus on the impact of the hardware part of the system[6].

Hardware architectures complexity is steadily increasing due to the massive integration of sophisticated functionalities, e.g. dynamic energy management. Due to the transistors integration limit and the thermal wall, a shift towards multi and many-cores architectures has been operated. These provide a considerable computation power, however they raise several problems for efficiently designing and implementing software on them. More importantly, they pose together with others hardware mechanisms such as caches, several challenges for the early estimation of systems performance. To illustrate these challenges, we discuss hereafter the impact of various hardware mechanisms on the system performance. We mainly focus on timing and energy aspects.

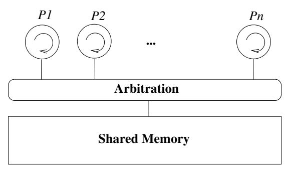

Fig. 1. Memory contention and arbitration.

Memory Contention. Consider the problem of characterizing the execution time or the consumed energy of a process P (executing some computation function, i.e. a set of sequential instructions) running on a specific processing unit. The latter is first assumed to exclusively execute P, i.e. no scheduling algorithm is required on top of it, and thus no context swapping overhead is induced for the

[6] Challeneges related to modeling the environment are out of the scope of this chapter.

moment. Data to be processed by P is stored in a shared memory which is accessible by different processing units executing similar tasks as shown in Figure 1. Thus, the execution time of P will vary depending on the number of processes and the way they are accessing the shared memory each time. Formally, if at time t, the processing unit executing P tries to access the memory alone, its execution time will be x_1, which is the time of executing the instructions of P when directly accessing the memory and getting the required data. Now, assume that at time $t' \neq t$, the processing unit executing P tries to access the memory simultaneously with $n-1$ others concurrent processing units (in competition). The execution time of P will be $x_2 = x_1 + \epsilon$, where ϵ is a time overhead encompassing the arbitration time and the waiting time (stall) to access the memory as illustrated in Figure 2. The overhead ϵ varies with respect to the number of concurrent processes simultaneously accessing the memory and the implemented arbitration policy. Hence, different executions of this setting provide different execution times of P.

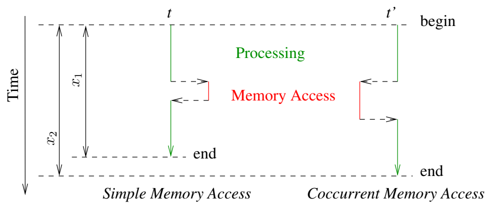

Fig. 2. Contention impact on execution time.

Memory Architectures. In the previous paragraph, we considered a relatively simple setting where the shared memory is designed as a single addressed space accessible by all the processing units in a similar way. With the progress of the hardware design, more involved memory architecture were proposed to overcome the memory access latency. It is known that the processing units are generally faster than the memory and thus their access represent a bottleneck for computation. Architectures such as the Tightly-Coupled Memory (TCM) or the Non-Uniform Memory Access (NUMA) aim at reducing the access latency by providing parallel access to different memory banks (different addressed spaces). The main idea behind these architecture is to divide the shared memory into different portions accessible in parallel (a portion for each processing unit), with a fast local access and slower distant one. The point is that such solution may reduce latency by exposing more parallelism for access. However, this should be carefully mitigated since the concurrency effect is still important as it depends on how the required data (by each process) is mapped onto the portions of the memory. Concretely, when the data required by two or more processes is mapped onto the same portion, a concurrent access will be necessary to fetch it. Thus, arbitration is still needed at this level. In these examples, note that it is increasingly hard to follow and understand the impact on the time and energy evolution of the system, although, we only considered data memory. Similar technologies are generally used to store instructions. Furthermore, other architectures propose structured memories with several levels and hence increasing the complexity of understanding and estimating the impact on the system performance.

Caches and DMAs. In the setting shown in Figure 1, assume that we introduce caches and/or DMAs. These hardware components will further impact the variation of the execution/communication times and similarly the energy of the system. For instance, for caches, depending on miss or hit situations, a processing unit will require to access the main (shared or local) memory or to get the data from the cache which is in turn a shared resource and requires arbitration when accessed concurrently. As for the DMA, its impact on the system performance can be interpreted as follow. The DMA enables the processing elements to perform data transfer in parallel to computation. Concretely, the processing unit only initiates the communication and another specialized hardware component will perform it while the processing unit continues its computation. When the transfer is done, mechanism such hardware interruptions is used to notify the processing element. This has the advantage to reduce communication overhead in the processing element, however it consumes more energy as it uses specialized hardware. Moreover, it makes the task of observing the exact communication and computation times more involved.

Multi-processing, Pipelining, and Others. In addition to the previous hardware functionalities, a processing element is generally able to execute several tasks in (pseudo-)parallel. This requires to use a scheduling policy to manage them (or synchronization/mutual exclusion mechanisms). A context swap between the executing processes (save/restore of stack) is thus operated, which generates an additional time and energy overhead. More complicated settings are induced by more sophisticated mechanism such as pipelines (with different levels), execution speculation mechanisms, and dynamic frequency scaling procedures. Techniques such as tasks migration, which moves processes from a processing element to another or onto a completely different cluster to equilibrate energy consumption or temperature over a chip/cluster add a huge complexity to the system.

Modeling the previously discussed hardware aspects (and many others which are not discussed here) in details is not possible neither recommended during the early stages of ES design. High-level models have to be abstract, easy to understand, and rapid to build and to analyze. As shown in the previous paragraphs, hardware aspect might be too complex to understand and to model in detail, especially for application development teams. These have as mission to design and implement applications which satisfy specific performance requirements in addition of being functionally correct. This should not be conditioned by a detailed comprehension and definitely not a detailed modeling of the functional and the non-functional behavior of target platforms. Nevertheless, performance aspects should be somehow taken into account early in the design to enable faithful modeling and later-on a trustworthy analysis towards good designs.

In the next section, we review some state of the art techniques for modeling performance of ES. We discuss their advantages and weaknesses and argue in favor of probabilistic techniques that we believe to be well-suited for characterizing ES performance faithfully.

3 Performance Modeling: State of the Art

In this section, we review methods and models used in the literature for ES performance characterization. The section will be organized around 3 important issues namely, (i) the used methods for gathering performance details during early design stages, (ii) given raw performance data, the usually utilized models to model performance, and finally, (iii) the methods used to build such models.

3.1 Techniques for Gathering ES Performance

State of the art techniques for gathering low-level performance information in early design phases can be classified into three different families. The first uses human expertise or documentation such as constructors data sheet of hardware architectures [25]. When available, this provides an easy way to get performance information, albeit it does not necessarily include details about the system performance e.g. caches impact. The second used technique is based on source/binary/object code, e.g. static analysis and code inspection [12, 8]. While it can give an idea on the expected system performance, this technique suffers from the absence of dynamic behavior of the system. The third technique is more accurate and is widely used, although the most time and resource consuming. It relies on executables models/implementations, i.e. high/low-level simulation or execution [37, 41, 49]. These techniques are not exclusive, that is they can be combined together during the same process

3.2 Characterizing ES Performance: Models and Methods

Detailed Representations A widely used technique in the context of ES design is to rely on a detailed description of the target hardware (a virtual platform, an RTL description, an FPGA implementation) and to perform co-simulation/emulation. That is, to simulate/emulate a model of the application on top of the hardware part of the system [48]. This enables to try different system configurations e.g. mapping, buffers sizes, and to access a detailed view of the system performance. Besides the issues related to building the detailed hardware description and the application implementation, additional technical issues are to handle such as measuring the performance and managing scalability issues, e.g. limited size of FPGAs. Moreover, an important time is necessary to perform co-simulation/emulation, e.g. simulating the boot of a Linux on an RTL description of a processor takes a half day. Such approach is empirical and provides no guarantee on the obtained performance. The reason is that the simulated/emulated artifact do not generally rely on a clear formal semantics, e.g. a SystemC description of the hardware architecture combined with a C/C++ implementation of the application. Hence, only classical testing [5] techniques are possible to check the performance requirements.

Abstract Representations As apposed to the previous procedure, the approach for formal modeling and evaluation of ES performance relies on mathematically sound models of the application and the target hardware. The first advantage of such approach is that it allows to evaluate the system performance early, i.e. before spending too much time building concrete implementations. Furthermore, it enables automatic code generation once the model is validated. The second advantage is that models validation relies on formal verification techniques which provide mathematical guarantees for performance evaluation results (in addition to functional requirements). However, as stated in the Section 2, it is challenging to build detailed hardware models (especially formal ones) during early design stages. Thus, abstract models are often built and combined with the application model in order to asses the expected ES performance formally. This approach requires to characterize the system performance, i.e. to build a parsimonious (abstract) formal representation of performance usually based on incomplete view of the system. Many approximation schemes providing different abstraction and faithfulness levels are possible.

A widely used class of techniques for characterizing performance especially timing uses upper/lower bounds. These techniques are mostly used for modeling and verifying hard real-time systems. They take their roots on well known theories such as the Queuing theory [24] and the Network Calculus [10]. They essentially reason on best/worst case scenarios and have as objective to build upper/lower bounds on performance. Worst Case Execution Time (WCET) analysis techniques [61] for instance are used compute the longest execution time in order to guarantee hard deadlines of executing tasks with respect to some scheduling policy. Such techniques are challenging and heavily dependent on the target hardware. Moreover, they are increasingly difficult and sometime impossible to apply in the case of multi and many-cores architectures [20].

Compositional analysis techniques have been proposed to handle the complexity of performance evaluation of heterogeneous real-time systems. These techniques mainly rely on the Real Time Calculus (RTC) method [56], which characterizes the input stream (workload) and the processing power of some task as arrival and service curves respectively. It then gives exact bounds on the output stream as a function of its input stream. The latter can be then used as input to the next component and so on. Arrival curves capture upper and lower bounds of arrival time of a class of input events, while service curve capture upper and lower bound of the processing time of consecutive events for any potential stream. RTC was adopted and extended in several researches such as SymTA/S (Symbolic Timing Analysis for Systems) [26], MAST [23], and the Modular Performance Analysis (MPA) [18]. The latter is used within the DOL methodology [58] for system-level performance analysis in the context of ES. As stated earlier, these methods are conceived to be conservative and thus are more appropriate to address stringent constraints such as hard real-time requirements. Consequently, they often result in over-dimensioned designs.

The previous methods generally produce analytical models and have the advantage of short analysis time. However, they generally fail to capture complex interactions and state-dependent behavior. The latter are enabled by another class of models known as operational models, e.g. timed automata [2], which enable a state-space representation of the system. Thus, they allow to build more accurate models. Nevertheless, these suffer from the state space explosion problem which results in important analysis time. An important advantage however is their ability to abstraction. Actually, it is possible using these models to represent the same functionality at different levels of details. Additional techniques for characterizing performance are also used in the literature. They often rely on simple averages or median measures [6, 41]. These provide too coarse characterization and thus are not very useful for building faithful models.

Probabilistic Representations Characterizing performance probabilistically avoids modeling sophisticated hardware functionalities in details. It enables capturing the gist of performance evolution and its associated fluctuation, e.g. the interference due to concurrent access to a shared resource is interpreted as a stochastic evolution over time. This provides a natural view for representing systems performance in a faithful and parsimonious way. For example, when tossing a coin, it is common to consider that its outcome is probabilistic, i.e. head and tail have equal probability to appear when assuming a fair coin. However, this is an abstract view of reality. Actually, if we are able to precisely characterize the coin, e.g., its weight, and it initial position, the environment where the experiment is happening, e.g., wind speed and direction, and the flipping power, we would be able to precisely compute the outcome of the experiment by applying the classical laws of physics.

Probabilistic modeling provides various abstraction possibilities, ranging from simple point estimate with confidence intervals, e.g. mean, variance, to more sophisticated models, e.g. probability density function, Markov Models. This depends on the level of details required to solve a given problem but also on the used method to build such models. Several techniques in the literature were extended to a probabilistic setting to take advantage of this view. Such as the Probabilistic WCET analysis techniques [20, 17] which enable to compute probabilistic upper bounds, that is a classical upper bound with the probability to exceed it, probabilistic real-time calculus [30, 53], probabilistic timed automata [34] and stochastic timed automata [4].

4 The *ASTROLABE* Approach

In this section, we review a recent work that illustrates our view of probabilistic performance characterization and how it may be used within a complete ES design process. The considered work introduces an approach for building and verifying faithful stochastic ES models that combines the functional behavior of the system with its performance information. We briefly overview the proposed approach and its different steps. Then, we discuss the assumptions it makes for inferring sound probabilistic performance models in order to propose later more general techniques.

4.1 Overview

The *ASTROLABE* approach was introduced in [45]. It targets the earliest phases of systems design and aims at providing fast and accurate quantitative evaluation of possible design alternatives with respect to performance requirements, e.g. time. As shown in Figure 3, the approach considers as inputs a parametric application model, a hardware architecture model, one or more mappings, i.e. a binding of the software in to hardware components, and a set of system requirements. The approach starts by automatically building a functional system model consisting of the combination of the software application with the hardware architecture with respect to the given mapping [11]. The input application and architecture models are purely functional and are expressed in the BIP formalism [7]. These may be obtained automatically through refinement from higher level specifications [8] or provided directly by the designer.

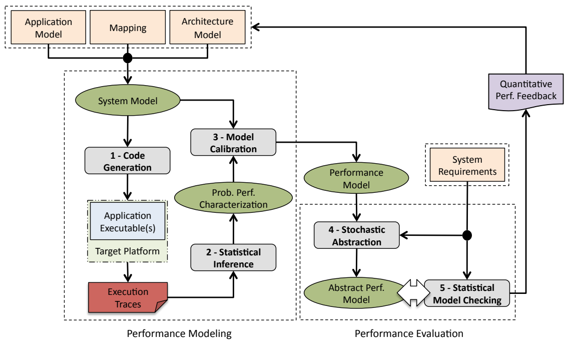

Fig. 3. Illustration of the *ASTROLABE* approach.

A complete iteration of the approach builds and analyzes a specific configuration, where all the application parameters and the mapping are fixed, against the given performance requirements. Quantitative results corresponding to different configurations enable to choose the most adequate and to refine the input models accordingly. To do so, the approach combines two activities, namely system modeling followed by performance evaluation. The former aims at producing a faithful model combining functional and performance information, while the latter aims at performing fast and accurate performance analysis. Both modeling and evaluation are composed of sub-tasks as shown in Figure 3.

The modeling phase consists of three successive steps: (1) A distributed implementation of the application and the the corresponding deployment code on the target platform are automatically generated given some runtime support. The code generation step produces instrumented code with respect to the input requirements which specify the performance aspects to characterize. (2) The generated implementation is executed on the target platform and the obtained traces are analyzed to build a probabilistic characterization of the specified performance aspects. This relies on a statistical inference procedure, namely distribution fitting described below. Finally, (3) a model calibration step integrates the obtained probabilistic characterization of performance into the functional system model, which produces a stochastic $\mathcal{S}$BIP model [44, 43].

In the other hand, the performance evaluation phase is two-fold: (4) a more compact and equivalent representation with respect to the original model is automatically built [46]. The goal of this step is to speed up the verification of performance requirements. (5) The analysis part is based on Statistical Model Checking algorithms [63, 27], which probabilistically answer if the system model satisfies the performance requirements up to a given precision.

The *ASTROLABE* approach is supported by a tool-flow that automates almost all the underlying steps and it was successfully applied to model and analyze a real-life case-study consisting of a distributed image recognition application running on many-cores platform [45].

4.2 Distribution Fitting

The distribution fitting procedure proposed in the *ASTROLABE* approach aims at fitting a probability distribution to performance observations obtained by executing an automatically generated application implementation on a target architecture. It consists of three successive steps which are briefly described in this section (for a complete description kindly refer to [45]).

The first step in this procedure is an exploratory analysis. It consists of verifying that the considered data is adequate for such procedure, i.e. that the performance observations are *independent and identically distributed (iid)*. In addition, this step tries to identify potential probability distribution families that potentially fit the data. The second step in the procedure consists of estimating the parameters of the identified probability distribution candidates. For this, it relies on well-known estimation procedure such as the Maximum Likelihood and the Moment Matching estimates [3, 19]. Finally, once the candidate distributions parameters are estimated, they are evaluated using goodness-of-fit tests to decide if they represent a good fit to the input data and to select the best fit. For this it uses test statistics such as the Kolmogorov-Smirnov (K-S), Anderson-Darling (A-D), and Carmer-Von Mises (C-VM) [60, 32]

4.3 Assumptions and Shortcomings

The *ASTROLABE* approach relies on a formal semantics in all its phases, which is important for a consistent design process. The distribution fitting procedure proposed to infer probabilistic models from concrete execution traces of the system represents a fundamental step to ensure a faithful modeling of performance. Furthermore, using abstraction and verification techniques for a quantitative performance evaluation provides fast and accurate analysis results.

Despite these advantages, in its current status, the approach have some weaknesses due to simplifying assumptions and restrictions which are necessary for the performance characterization proposal to work. These assumptions are verified for specific classes of systems where the performance observations satisfy the *iid* assumption. To ensure this assumption, the input models to the approach are restricted to the following:

– The application model is required to follow a process network model of computation, i.e. a set of computation processes (each operating sequentially) coordinated through communication channels. Such models enforce a clear separation between computation and communication.
– The hardware architectures model correspond to many-cores platforms with homogeneous processing elements sharing a main memory. Processing cores and memory may be organized in clusters, i.e. each cluster has a set of processing elements and a local memory. Clusters may communicate through a NoC or a bus. The considered platforms do not use sophisticated hardware functionalities such as caches, or pipelining. Thus, the main source of interference at this level is the contention on shared resources.
– The considered mappings statically bind each computation processes to a processing core, i.e. no multi-threading on cores is allowed. Communication channels are mapped into shared memory.

5 Performance Modeling using Probabilistic Models

Probabilistic modeling usually consists of manually deriving a probabilistic characterization from given specifications of some artifact. In some cases, we are only given output data of an experiment e.g. data collected during a poll. Such data represents a partial view of the artifact, that is, in statistical terms, a sample of the complete data population. In such situations, inferential analysis, that is a bottom up path, is used to build the model. Statistical inference is thus the process of probabilistically characterizing the behavior of an artifact based on partial observations. Such approach is often denoted model fitting [36, 60].

In our context we are concerned with performance measurements obtained by simulating/executing embedded software implementations on a hardware architecture. The goal is to characterize the system performance by finding an appropriate model that faithfully and parsimoniously describe it. To do so, various probabilistic models exist, spanning from standard probability distributions, e.g., Exponential or Normal, to more complex ones such as regression models.

From this perspective, data is assumed to be generated by a stochastic process (Definition 1) for which the governing law is unknown. The goal of the statistical inference is to infer such a law from given observations which represent a partial view of the process since the whole population is generally not available. Formally, given $x_1, ..., x_n$ a set of observations, there exist $X_1, ..., X_n$ random variables such that x_i is a possible realization of X_i, for $1 \leq i \leq n$. The set of random variables $X_1, ..., X_n$ represent together a stochastic process.

Definition 1 (Stochastic Process). *Given two sets S and T, a stochastic process $\mathcal{X}$ is a collection of random variables $\{X_i \mid i \in T\}$ where each X_i is an S-valued random variable. The set S is called the state space of the process.*

In the context of performance modeling, the random variables $X_1, ..., X_n$ represent measurements of the same performance metric, say the execution time of a computation process, at different time points, i.e. the indexes $i \in T$ actually represent the measurement times. They typically correspond to the set of non-negative integers $\mathbb{Z}^*$ in the case of discrete-time stochastic processes and to the set of non-negative real number $[0, \infty)$ for continuous-time stochastic processes.

Depending on the target probabilistic model and on the underlying data, different inference methods can be used to construct the desired model. In the *ASTROLABE* approach for instance, we targeted a simple probabilistic model consisting of standard probability distributions as discussed in the previous section. In that work, it is assumed that performance observations are *independent and identically distributed*. *Independent* means that given the realization x_i of the random variable X_i, it does not provides any information about the realization x_j of X_j, for $i < j \leq n$, Formally $P(X_j = x_j \mid X_i = x_i) = P(X_j = x_j)$. *Identically distributed* states that the underlying random variables follow a same probability distribution $D(\omega)$, where $\omega \in \Theta$ is the set of parameters of the distribution defined over the space Θ.

This *iid* assumption may be seen as an abstraction choice. That is, since the underlying random variables are independent and they follow the same distribution $D(\omega)$, they can be modeled by a single random variable X that follows the distribution $D(\omega)$ and where $x_i, \ldots, x_n$ are possible realizations of X with probabilities defined by $D(\omega)$. In this view, one can forget about the time dimension of the stochastic process, i.e. the measurements order, as it is assumed to has no impact on the process evolution. While providing a nice abstract and parsimonious model of performance, the *iid* assumption is usually too strong. In practice, measurements performed on real-life systems relying on sophisticated hardware functionalities do not generally satisfy it.

Consider a hardware architecture implementing functionalities such as the ones presented in Section 2. Such hardware introduces a dependency between the random variables $X_1, \ldots, X_n$ representing the performance metric. Concretely, we can write that $P(X_i = x_i) = P(X_i = x_i \mid X_{i-1} = x_{i-1}, \ldots, X_0 = x_0)$. Assume we are measuring the execution time of some computation process at different times. The value obtained at time i may impact future measures because of the effects of caches, branch prediction, or pipelining. Such hardware components are known to introduce a history within the data evolution [62]. In general, they may create additional dependency with other random variables modeling the execution context. For instance, the execution time of a process depends on the data cache status (hit or miss), the number of concurrent processes accessing a shared resource, the used arbitration policy, and other factors. Moreover, the impact of such components is not always the same, e.g. a cache hit does not systematically implies a lower execution time. This kind of behavior is known as timing anomaly in the literature [52].

Capturing such a history or dependency is important to characterize performance faithfully. However, the obtained models must be parsimonious and concise to enable fast analysis. Ideally, they should be obtained automatically. In the following we first present some models that we believe promising for ensuring such requirements. Next, we survey some techniques that enable to obtain such models automatically from measurements.

5.1 Probabilistic Models

The distribution fitting procedure presented in [45] is one among various model fitting procedures [36, 60]. It aims at inferring a probability distribution as a target performance model. Different probabilistic models may be used and they imply to use different techniques with different complexities. The chosen model may or may not take into account the internal structure of the data induced by the hardware part of the system. This primarily depends on the required level of abstraction, but also on the inherent properties of the data. In the following, we present three models that capture the data internal structure in different ways.

Mixture Distributions. Mixture distributions [40, 47] are useful when the considered data provides clues to be drawn from different parent populations. In practice this may corresponds to performance measurements performed in different computation modes, e.g. the energy consumption measured during a power-save mode and a normal mode. The decision of fitting such a model to the data usually follows an exploratory analysis (similar to the distribution fitting procedure in [45]). For instance, a histogram of the data may show a clear multi-modality[7] as illustrated in Figure 4 which shows a bimodal shape.

A mixture distribution enables to capture data sub-populations and to fit it with a single probability distribution that can be used to perform classical computation, e.g. expectations. Let $f_1(x), \ldots, f_m(x)$ be a set of probability density functions associated with distributions $D_1(\omega), \ldots, D_m(\omega)$, and let $w_1, \ldots, w_m$ be some weights such that $\sum_{i=1}^{m} w_i = 1$ (they follow an additional probability distribution which we denote the selection distribution W). Then, the mixture distribution density function f is defined as $\sum_{i=1}^{m} w_i f_i$. An important assumption in this context is the independence between the selection distribution W and the underlying distributions f_i. That is, we assume that a component distribution is first selected randomly according to W, then the selected distribution is used to draw a value. Figure 4 shows an example data illustrated using a histogram with a clear bimodality. The figure also shows the fitted mixture distribution composed of two Normal distributions [9]. It is worth mentioning that we are not aware of works that uses mixture distributions to characterize performance of ES.

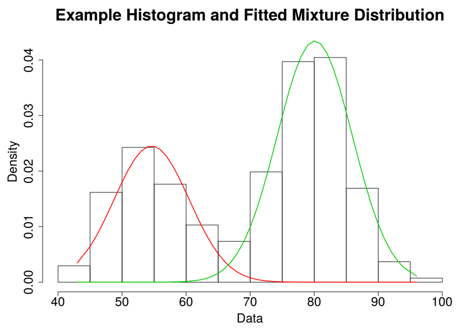

Fig. 4. A Mixture Distribution Example.

[7] A mode can be identified as a prominent peak in the histogram.

Regression Models. Regression Models provide a mathematical tool to understand, explain, and forecast on data. They enable to model the underlying stochastic process evolution as depending on internal or external factors. Hence, they allow for capturing dependencies within the process or with respect to other impacting factors. More precisely, the process evolution (denoted response or output variable) can be explained by its own history evolution or by other random variables (denoted explaining variables). In our context, we focus on the first case, that is we provide models that characterize the process evolution in term of its own history. The second case is equally interesting for investigation in this context as it enables to characterize a given performance metric as a function of its execution context e.g. caches status, shared resource contention.

In general, regression models can be linear (respectively non-linear), that is the response variable is a linear (respectively non-linear) combination of the explaining variables. Furthermore, different regression models can be used to model stochastic processes with different properties. In the following, we focus on regression models for stationary stochastic processes [14, 42]. That is the process has a constant variance over time (no trend), a constant autocorrelation structure over time, and no periodic fluctuation (no seasonal behavior). We believe that this class of process may be interesting for ES performance measurements.

There exist various regression models that can be used to characterize stationary processes. A well-known class is the *Autoregressive Models* (AR) which are linear regression models that represent the response variable as a linear function of previous steps (explaining variables) in addition to a random error.

$$X_i = c + \theta_1 X_{i-1} + \cdots + \theta_p X_{i-p} + \epsilon_i = c + \epsilon_i + \textstyle\sum_{j=1}^{p} \theta_i X_{j-1}$$

where c is a constant, $\theta_i, \ldots, \theta_p$ are parameters of the model, and p is the order of the AR model, i.e. the number of considered explaining variables (how many previous steps impact the next one). Another commonly used regression model for stationary processes is the *Moving Average* (MA). The latter is also a linear regression model. However, it represents the response variable as a function of random error terms evolving around the expected value (the mean) of the data.

$$X_i = \mu + \epsilon_i + \varphi_1 \epsilon_{i-1} + \ldots + \varphi_q \epsilon_{i-q} = \mu + \epsilon_i + \textstyle\sum_{k=1}^{q} \varphi_i \epsilon_{j-1}$$

where μ is the expectation of X_i, $\varphi_i, \ldots, \varphi_q$ are parameters of the model, and q is the order of the model. In both models, ϵ represent random error terms which are often assumed to be independent and identically distributed random variables following a Normal distribution $N(0, \sigma^2)$, where σ^2 is its variance. In addition to the previous models a combination of them can be also used to characterize stationary stochastic processes. This is known as Box-Jenkins ARMA model [13, 14] and includes p autoregressive terms and q moving average ones.

$$X_i = c + \epsilon_i + \textstyle\sum_{j=1}^{p} \theta_i X_{j-1} + \sum_{k=1}^{q} \varphi_i \epsilon_{j-1}$$

It is also possible to handle non stationary processes by differencing the process one or more times to achieve stationarity. This leads to the so called ARIMA models, where the "I" stand for integrated [42].

A few works in the literature have used regression models to characterize performance of embedded systems. For example, in [22], regression models corresponding to processes execution time over a single processor are used to calibrate high-level behavioral models in the context of the the VCC system-level modeling and analysis methodology. In [1], a tool for power estimation on co-processors is presented, it also relies on regression techniques to learn predictive power consumption models as functions of different aspects of the system.

Markov Models. Markov models or chains are stochastic processes that have a particular properties known as the Markov property. Concretely, they are memoryless processes, in the sense that the probability of reaching the next state of the system only depends on its current state and not on the complete history of the process. Formally, we write that:

$$P(X_{i+1} = x_{i+1} \mid X_i = x_i, \cdots, X_0 = x_0) = P(X_{i+1} = x_{i+1} \mid X_i = x_i)$$

A Markov chain is often represented as a directed graph where vertices represent the model states and the edges represent transitions from one state to another. An edge from a state s_i to a state s_{i+1} is labeled with the probability $P(X_{i+1} = x_{i+1} \mid X_i = x_i)$. Figure 5 illustrate an example of a Markov chain modeling the behavior of the Craps Gambling game [5]. In this game, a player starts by rolling two fair six-sided dice. The outcome of the two dice determines whether he wins or not. If the outcome is 7 or 11, the player wins. If the outcome is $2, 3$, or 12, the player looses. Otherwise, the dice are rolled again taking into account the previous outcome (called point). If the new outcome is 7, the player looses. If it is equal to point, he wins. For any other outcome, the dice are rolled again and the process continues until the player wins or looses.

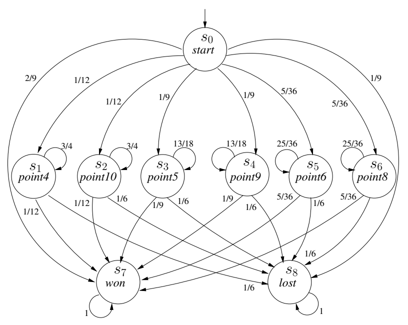

Fig. 5. A Markov Chain Example.

Markov models are by far the most used to characterize ES performance [34, 15, 33]. However, most of these works start from systems specifications to manually build the Markov model. Only few works follow an inferential path, i.e. uses concrete execution data to obtain a faithful model [38, 39].

5.2 Learning and Fitting Techniques

In this section we survey some techniques that may be used to automatically build the models presented above. These techniques mainly rely on model fitting which is represent an important part of the family of machine learning techniques. The latter is an active field of research and learning algorithms are constantly developed and improved in order to address new challenges and new classes of problems (see [59] for a recent survey).

Fitting Mixture Models. To estimate the parameters of mixture models, i.e. the parameters ω of the sub-distributions D_i and the selection distribution W, one may use classical estimation techniques such as the likelihood estimate or the moment matching estimate [3, 19]. However, in this setting such techniques often fail to perform the estimation. Generally, for fitting a mixture model to input data, we rely on a well-known algorithm, namely the Expectation Maximization (EM) [14, 40]. This is an iterative algorithm used for computing the maximum likelihood estimate when the considered data is incomplete. Some tools also exist for automatically fitting such models [9].

Fitting Regression Models. Fitting an ARMA model to the data can be done using classical least square regression after fixing the orders p and q [42]. This will estimate the values of the model parameters by minimizing the error term. The fitting procedure follows usually three main steps, namely model identification, that is to check if the considered data is stationary and if there is a seasonality that need to be taken into account (one can use run plot or autocorrelation plot to identify it). This first step is also useful to identify p and q. The second step is the model estimation, that is estimating the parameters of the model. As stated earlier one can relies on classical estimation approaches such as least squares and maximum of likelihood [3, 19]. The last step of the procedure is an evaluation step. It consists of verifying that the model residuals are independent drawings from a fixed distribution [42]. Recently, a new and rich class of models and techniques were proposed in this context. They are known state-space models and were originally developed in the context of linear systems control [14]. They rely for instance on Kalman recursions and EM algorithm [14].

Learning Markov Models. In the last years, several works were proposed to learn Markov Models with their different varieties. Aalergia [38] is a state merging algorithm that enable to learn deterministic Markov Chain. It is a variant of the alergia algorithm proposed by Carrasco and Oncina in early nineties [16]. Given a sample of execution traces, the algorithm builds an intermediate graph that represents all the traces in the input sample and their corresponding frequencies. Then, it iteratively merges the nodes of the graph that have the same labels and similar probability distributions until reaching a compact version, which corresponds to the final Markov chain. An extension of this algorithm was also proposed in [39] to enable learning Markov models with non-determinism, namely Markov Decision Processes (MDPs). Other techniques based on the Bayesian approach were proposed for learning the parameters and the structure of Hidden Markov Models (HMMs) [51]. They are also following a state merging procedure [55]. The previously mentioned works essentially focus on discrete-time models. Other works, albeit few, exist for learning continuous-time Markov models. For instance, in [54] it is proposed to learn continuous-time Markov chains (CTMCs) models from sample traces following the state merging approach by providing a variant of the alergia algorithm. In [21], an algorithm that follows the same strategy is proposed to learn more general continuous-time models, namely General Semi-Markov Processes (GSMPs).

5.3 Perspectives

In the previous sections, we focused on presenting models and techniques for characterizing performance probabilistically. Nonetheless, given such characterization, an important question remains on how to combine the performance characterization with the function behavior of the system within a single formal framework. Within the *ASTROLABE* approach, a calibration procedure is used in order to augment functional BIP models with performance characterization in form of probability distributions. This procedure produces stochastic $\mathcal{S}$BIP models [43, 44].

We speculate that a similar procedure is potentially applicable when charactering performance as mixture distributions since they are formally described a probability distribution. On the other side, this might be more complicated for regression and Markov models. The latter might be easier to integrate within BIP models due to their underlying operational semantics. In [43], a transformation of a Markov chain to a stochastic $\mathcal{S}$BIP component is formally defined. However, integrating it within the functional model requires to add the appropriate glue. Regression models are more involved because of their underlying analytical models. They require more investigation as how to combine them with operational functional models, e.g. BIP, and about the semantics it will induce.

References

1. Sumit Ahuja, Deepak A Mathaikutty, Avinash Lakshminarayana, and Sandeep K Shukla. Scope: Statistical regression based power models for co-processors power estimation. *Journal of Low Power Electronics*, 5(4):407–415, 2009.
2. Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, April 1994.
3. T.W. Anderson and J.D. Finn. *The New Statistical Analysis of Data.* Springer, 1996.
4. C. Baier, N. Bertrand, P. Bouyer, T. Brihaye, and M. Grösser. Probabilistic and topological semantics for timed automata. In *Proceedings of the 27th International Conference on Foundations of Software Technology and Theoretical Computer Science*, FSTTCS, pages 179–191, Berlin, Heidelberg, 2007. Springer-Verlag.
5. C. Baier and J.P. Katoen. *Principles of Model Checking.* MIT Press, 2008.
6. Amol Bakshi, Viktor K Prasanna, and Akos Ledeczi. MILAN: A model based integrated simulation framework for design of embedded systems. *ACM Sigplan Notices*, 36(8):82–93, 2001.
7. Ananda Basu, Bensalem Bensalem, Marius Bozga, Jacques Combaz, Mohamad Jaber, Thanh-Hung Nguyen, and Joseph Sifakis. Rigorous component-based system design using the BIP framework. *IEEE Softw.*, 28(3):41–48, May 2011.
8. Ananda Basu, Saddek Bensalem, Marius Bozga, Paraskevas Bourgos, Mayur Maheshwari, and Joseph Sifakis. Component assemblies in the context of manycore. In *Formal Methods for Components and Objects*, pages 314–333. Springer, 2013.

9. Tatiana Benaglia, Didier Chauveau, David R. Hunter, and Derek S. Young. mixtools: An R Package for Analyzing Finite Mixture Models. *Journal of Statistical Software*, 32(6):1–29, October 2009.
10. J-Y. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag, Berlin, Heidelberg, 2001.
11. Paraskevas Bourgos. *Rigorous Design Flow for Programming Manycore Platforms*. PhD thesis, Grenoble University, 2013.
12. Paraskevas Bourgos, Ananda Basu, Marius Bozga, Saddek Bensalem, Joseph Sifakis, and Kai Huang. Rigorous system level modeling and analysis of mixed HW/SW systems. In *MEMOCODE*, pages 11–20, 2011.
13. G.E.P. Box, G.M. Jenkins, and G.C. Reinsel. *Time Series Analysis: Forecasting and Control*. Forecasting and Control Series. Prentice Hall, 1994.
14. P.J. Brockwell and R.A. Davis. *Introduction to Time Series and Forecasting*. Number v. 1 in Introduction to Time Series and Forecasting. Springer, 2002.
15. Peter E. Bulychev, Alexandre David, Kim Guldstrand Larsen, Marius Mikucionis, Danny Bøgsted Poulsen, Axel Legay, and Zheng Wang. UPPAAL-SMC: statistical model checking for priced timed automata. In *Proceedings 10th Workshop on Quantitative Aspects of Programming Languages and Systems, QAPL 2012, Tallinn, Estonia, 31 March and 1 April 2012.*, pages 1–16, 2012.
16. Rafael C. Carrasco and José Oncina. Learning stochastic regular grammars by means of a state merging method. In *ICGI*, pages 139–152, 1994.
17. F. J. Cazorla, E. Quinones, T. Vardanega, L. Cucu-Grosjean, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim. PROARTIS: Probabilistically Analysable Real-Time Systems. Research Report RR-7869, INRIA, January 2012.
18. Samarjit Chakraborty, Simon Künzli, and Lothar Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *DATE*, volume 3, page 10190. Citeseer, 2003.
19. Glen Cowan. *Statistical Data Analysis*. Oxford University Press, Oxford, 1998.
20. Liliana Cucu-Grosjean, Luca Santinelli, Michael Houston, Codé Lo, Tullio Vardanega, Leonidas Kosmidis, Jaume Abella, Enrico Mezzeti, Eduardo Quinones, and Francisco J. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. In *the 24th Euromicro Conference on Real-Time Systems*, Pise, Italy, July 2012.
21. André de Matos Pedro, Paul Andrew Crocker, and Simão Melo de Sousa. Learning stochastic timed automata from sample executions. In *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*, pages 508–523. Springer, 2012.
22. P. Giusto, G. Martin, and E. Harcourt. Reliable estimation of execution time of embedded software. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '01, pages 580–589. IEEE Press, 2001.
23. M González Harbour, JJ Gutiérrez García, JC Palencia Gutiérrez, and JM Drake Moyano. Mast: Modeling and analysis suite for real time applications. In *13th Euromicro Conference on Real-Time Systems*, pages 125–134. IEEE, 2001.
24. D. Gross, J.F. Shortle, J.M. Thompson, and C.M. Harris. *Fundamentals of Queueing Theory*. Wiley Series in Probability and Statistics. Wiley, 2011.
25. Wolfgang Haid, Matthias Keller, Kai Huang, Iuliana Bacivarov, and Lothar Thiele. Generation and calibration of compositional performance analysis models for multi-processor systems. In *ICSAMOS*, pages 92–99, 2009.

26. Rafik Henia, Arne Hamann, Marek Jersak, Razvan Racu, Kai Richter, and Rolf Ernst. System level performance analysis - the SymTA/S approach. In *IEEE Proceedings Computers and Digital Techniques*, 2005.
27. T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate Probabilistic Model Checking. In *VMCAI*, pages 73–84, January 2004.
28. K. Huang, W. Haid, I. Bacivarov, M. Keller, and L. Thiele. Embedding formal performance analysis into the design cycle of MPSoCs for real-time streaming applications. *ACM Trans. Embed. Comput. Syst.*, 11(1):8:1–8:23, April 2012.
29. Z. J. Jia, A. Núñez, T. Bautista, and A. D. Pimentel. A two-phase design space exploration strategy for system-level real-time application mapping onto MPSoC. *Microprocess. Microsyst.*, 38(1):9–21, February 2014.
30. Y. Jiang and Y. Liu. *Stochastic Network Calculus*. Springer London, 2008.
31. Kurt Keutzer, Sharad Malik, Senior Member, A. Richard Newton, Jan M. Rabaey, and A. Sangiovanni-vincentelli. System-level design: Orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19:1523–1543, 2000.
32. Stuart A Klugman, Harry H Panjer, and Gordon E Willmot. *Loss models: from data to decisions*, volume 715. John Wiley & Sons, 2012.
33. M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic verification of hermans self-stabilisation algorithm. *Formal Aspects of Computing*, 24(4):661–670, 2012.
34. Marta Z. Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theor. Comput. Sci.*, 282(1):101–150, 2002.
35. K. Lampka, G. Giannopoulou, R. Pellizzoni, Z. Wu, and N. Stoimenov. A formal approach to the wcrt analysis of multicore systems with memory contention under phase-structured task sets. *Real-Time Syst.*, 50(5-6):736–773, November 2014.
36. Jean-Yves Le Boudec. *Performance Evaluation of Computer and Communication Systems*. EPFL Press, Lausanne, Switzerland, 2010.
37. Paul Lieverse, Pieter Van Der Wolf, Kees Vissers, and Ed Deprettere. A methodology for architecture exploration of heterogeneous signal processing systems. *Journal of VLSI signal processing systems for signal, image and video technology*, 29(3):197–207, 2001.
38. H. Mao, Y. Chen, M. Jaeger, T. D. Nielsen, K. G. Larsen, and B. Nielsen. Learning probabilistic automata for model checking. In *QEST*, pages 111–120, 2011.
39. Hua Mao, Yingke Chen, Manfred Jaeger, Thomas D Nielsen, Kim G Larsen, and Brian Nielsen. Learning markov decision processes for model checking. *arXiv preprint arXiv:1212.3873*, 2012.
40. N. Matloff. *From Algorithms to Z-Scores: Probabilistic and Statistical Modeling in Computer Science*. University Press of Florida, 2009.
41. S. Mohanty and V. K. Prasanna. Rapid system-level performance evaluation and optimization for application mapping onto SoC architectures. In *ASIC/SOC Conference, 2002. 15th Annual IEEE International*, pages 160–167. IEEE, 2002.
42. NIST/SEMATECH. *NIST/SEMATECH e-Handbook of Statistical Methods*. 2016.
43. Ayoub Nouri. *Rigorous System-level Modeling and Performance Evaluation for Embedded System Design*. Theses, Université Grenoble Alpes, April 2015.
44. Ayoub Nouri, Saddek Bensalem, Marius Bozga, Benoît Delahaye, Cyrille Jégourel, and Axel Legay. Statistical model checking QoS properties of systems with SBIP. *STTT*, 17(2):171–185, 2015.
45. Ayoub Nouri, Marius Bozga, Anca Molnos, Axel Legay, and Saddek Bensalem. Building faithful high-level models and performance evaluation of manycore embed-

ded systems. In *Twelfth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE), 2014*, pages 209–218. IEEE, 2014.

46. Ayoub Nouri, Balaji Raman, Marius Bozga, Axel Legay, and Saddek Bensalem. Faster statistical model checking by means of abstraction and learning. In *Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings*, pages 340–355, 2014.
47. R. Pearson. *Exploring Data in Engineering, the Sciences, and Medicine.* USA, 2011.
48. Andy D. Pimentel. The artemis workbench for system-level performance evaluation of embedded systems. *Int. J. Embedded Systems*, 2005.
49. Andy D Pimentel, Cagkan Erbas, and Simon Polstra. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *Computers, IEEE Transactions on*, 55(2):99–112, 2006.
50. Andy D. Pimentel, Mark Thompson, Simon Polstra, and Cagkan Erbas. Calibration of abstract performance models for system-level design space exploration. *J. Signal Process. Syst.*, 50(2):99–114, February 2008.
51. Lawrence Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
52. Jan Reineke, Björn Wachter, Stephan Thesing, Reinhard Wilhelm, Ilia Polian, Jochen Eisinger, and Bernd Becker. A definition and classification of timing anomalies. In *6th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis, July 4, 2006, Dresden, Germany*, 2006.
53. Luca Santinelli and Liliana Cucu-Grosjean. Toward probabilistic real-time calculus. *SIGBED Rev.*, 8(1):54–61, March 2011.
54. K. Sen, M. Viswanathan, and G. Agha. Learning continuous time markov chains from sample executions. In *First International Conference on the Quantitative Evaluation of Systems QEST.*, pages 146–155. IEEE, 2004.
55. A. Stolcke and S. Omohundro. Hidden markov model induction by bayesian model merging. *Advances in neural information processing systems*, pages 11–11, 1993.
56. L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *ISCA*, volume 4, pages 101 –104 vol.4, 2000.
57. L.0 Thiele, L. Schor, I. Bacivarov, and H. Yang. Predictability for timing and temperature in multiprocessor system-on-chip platforms. *ACM Transactions on Embedded Computing Systems (TECS) - Special section on ESTIMedia12, LCTES11, rigorous embedded systems design, and multiprocessor*, 12, Mar 2013.
58. Lothar Thiele, Iuliana Bacivarov, Wolfgang Haid, and Kai Huang. Mapping applications to tiled multiprocessor embedded systems. In *ACSD*, 2007.
59. Sicco Verwer, Rémi Eyraud, and Colin de la Higuera. Results of the pautomac probabilistic automaton learning competition. In *ICGI*, pages 243–248, 2012.
60. David Vose. *Risk analysis : a quantitative guide.* Wiley, 2008.
61. R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem&mdash;overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):36:1–36:53, May 2008.
62. Reinhard Wilhelm, Daniel Grund, Jan Reineke, Marc Schlickling, Markus Pister, and Christian Ferdinand. Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 28(7):966–978, 2009.
63. H. L. S. Younes. *Verification and Planning for Stochastic Processes with Asynchronous Events.* PhD thesis, Carnegie Mellon, 2005.