Feasibility Study of Probabilistic Timing Analysis Methods for SDF Applications on Multi-Core Processors

Ralf Stemmer, Hai-Dang Vu, Maher Fakih, Kim Grüttner, Sébastien Le Nours, Sébastien Pillement

Feasibility Study of Probabilistic Timing Analysis Methods for SDF Applications on Multi-Core Processors

Ralf Stemmer*, Hai-Dang Vu†, Maher Fakih‡, Kim Grüttner‡,
Sebastien Le Nours† and Sebastien Pillement†
* University of Oldenburg, Germany, Email: ralf.stemmer@uol.de
† University of Nantes, France, Email: {hai-dang.vu, sebastien.le-nours, sebastien.pillement}@univ-nantes.fr
‡ OFFIS e.V., Germany, Email: {maher.fakih, kim.gruettner}@offis.de

***Abstract*—Early validation of software running on multi-core platforms is fundamental to guarantee functional correctness and that real-time constraints are fully met. In the domain of timing analysis of multi-core systems, existing simulation-based approaches and formal mathematical methods are hampered with scalability problems. In this context, probabilistic simulation techniques represent promising solutions to improve scalability of analysis approaches. However, creation of probabilistic SystemC models remains a difficult task and is not well supported for multi-core systems. In this technical report, we present a feasibility study of probabilistic simulation techniques considering different levels of platform complexity. The evaluated probabilistic simulation techniques demonstrate good potential to deliver fast yet accurate estimations for multi-core systems.**

I. Introduction

Because of the growing demand in computational efficiency, more and more embedded systems are designed on multi-core platforms. Multi-core platforms are a combination of computation resources (general-purpose processors, specialized processors, dedicated hardware accelerators), memory resources (shared memories, cache memories), and communication resources (bus, network on chip). Compared to single processor platforms, the sharing of resources in multi-core platforms leads to complex interactions among components that make validation of software and estimation of extra-functional properties very challenging.

System level design approaches have been proposed to allow performance estimation of hardware/software architectures early in the design process. In system level design approaches, workload models are used to capture the influence of application execution on platform resources. Timing properties of architectures and related power consumption can then be assessed under different working scenarios. However, the efficiency of system level design approaches strongly depends on the created HW/SW architecture models that should deliver both reduced analysis time and good accuracy. Especially, captured workload models should correctly abstract low-level details of system components but still provide good estimates about the whole system performance. High-level component models must capture the possible variability in multi-core platform resources usage caused by cache management, bus interleaving, and memory contention. Creation of efficient architecture models represents thus a time-consuming effort that limits the efficiency of current system level approaches. Because of the growing complexity of architectures, existing real-time analysis methods, *i.e.*, simulation-based and formal mathematical approaches, show limitations to deliver fast yet accurate estimation of timing properties. Novel analysis techniques are thus expected to facilitate the evaluation of multi-core architectures with good accuracy and in reasonable times.

Probabilistic models represent a possible solution to capture variability caused by shared resources on parallel software execution [1]. Quantitative analysis of probabilistic models can then be used to quantify the probability that a given time property is satisfied. Numerical approaches exist that compute the exact measure of the probability at the expense of time-consuming analysis effort. Another approach to evaluate probabilistic models is to simulate the model for many runs and monitor simulations to approximate the probability that time properties are met. This approach, which is also called Statistical Model Checking (SMC), is far less memory and time intensive than probabilistic numerical methods and it has been successfully adopted in different application domains [2]. In the field of embedded system design, executable specifications built with the use of the SystemC language are now widely adopted [3]. SystemC models used for the purpose of timing analysis typically capture workload models of the application mapped on shared computation and communication resources of the considered platform. Timing annotations are commonly expressed as average values or intervals with estimated best case and worst case execution times. The adoption of SMC techniques to analyze SystemC models of multi-core systems is promising because it could deliver a good compromise between accuracy and analysis time, yet it requires a more sophisticated timing model based

This work has been partially sponsored by the DAAD (PETA-MC project under grant agreement 57445418) with funds from the Federal Ministry of Education and Research (BMBF). This work has also been partially sponsored by CampusFrance (PETA-MC project under grant agreement 42521PK) with funds from the French ministry of Europe and Foreign Affairs (MEAE) and by the French ministry for Higher Education, Research and Innovation (MESRI)

on probability density functions, inferred from measurements on a real prototype. Thus, the creation of trustful probabilistic SystemC models is challenging. Since SMC methods have rarely been considered to analyze timing properties of applications mapped on multi-core processors with complex hierarchy of shared resources, exploring their application on trustful probabilistic SystemC models remains a significant research topic.

In this technical report, we present a feasibility study of SMC methods for multi-core systems. The novelty of the established framework is twofold. The first contribution deals with the modeling process, including a measurement-based approach, to appropriately prepare timing annotations and calibrate SystemC models. The second contribution is about the evaluation of SMC methods efficiency with respect to accuracy and analysis time. Evaluation is done by comparing a real multi-core implementation with related estimation results. To restrict the scope of our study, we have considered applications modeled as Synchronous Data Flow Graphs (SDFGs). We have evaluated our framework on a Sobel filter case study. Two configurations of the multi-core processor with different levels of complexity are considered to analyze the relevance and effectiveness of SMC methods.

This technical report is organized as follows. In Section II we provide an overview and discussion of relevant related work. In Section III we provide background information on our chosen SDF model, considered multi-core timing effects and the applied SMC methods. Section IV presents the established modeling and analysis approach. The experimental results are described in Section V. Section VI discuss the benefits and limitations of the presented approach.

II. Related Work

A. Timing analysis approaches

Timing analysis approaches are commonly classified as (1) simulation-based approaches, which partially test system properties based on a limited set of stimuli, (2) formal approaches, which statically check system properties in an exhaustive way, and (3) hybrid approaches, which combine simulation-based and formal approaches.

Simulation-based approaches: Different simulation-based approaches have been proposed to evaluate multi-core architecture performance early in the design process. In the proposed approaches, models of hardware-software architectures are formed by combining an application model and a platform model. In the early design phase, full description of application functionalities is not mandatory and workload models of the application are used. A workload model expresses the computation and communication loads (e.g., time, power consumption, memory cost) that an application causes when executed on platform resources. Captured performance models are then generated as executable descriptions and simulated. The execution time of each load primitive is approximated as a delay, which are typically estimated from measurements on real prototypes or analysis of low level simulations. SystemCoDesigner [4], Daedalus [5], SCE [6], and Koski [7] are good examples of academic approaches; they are compared in [8]. Other existing academic approaches are presented by Kreku et al. in [9] and by Arpinen et al. in [10]. An overview and classification of the considered related work can be found in Table I. Moreover, some industrial frameworks such as Intel CoFluent Studio [11], Timing Architect [12], ChronSIM [13], TraceAnalyzer [14] Space Codesign [15] and Visualsim from Mirabilis Design [16] have emerged also.

Simulation-based approaches require extensive architecture analysis under various possible working scenarios but created architecture models can hardly be exhaustively simulated. Due to insufficient corner case coverage, simulation-based approaches are thus limited to determine guaranteed limits about system properties. One other important issue concerns the accuracy of created models. As architecture components are modeled as abstractions of low level details, there is no guarantee that the created architecture model reflects with good accuracy the whole system performance. Finally, with the rising complexity of many-core platforms, execution of simulation models requires more simulation time.

Formal approaches: Due to insufficient corner case coverage, simulation-based approaches are limited to determine guaranteed limits about system properties. Different formal approaches have thus been proposed to analyze multi-core systems and provide hard real-time and performance bounds. These formal approaches are commonly classified as state-based approaches and analytical approaches.

Most of the available static real-time methods are of analytical nature (c.f. [17] for an overview). Since analytical methods depend on solving closed-form equations (characterizing the system temporal behavior), they have the advantage of being scalable to analyze large-scale systems. MPA-RTC [18] and SymTA/S [19] are two representatives of compositional analytical methods for multi-core systems.

Despite their advantages of being scalable, analytical methods abstract from state-based modus operandi of the system under analysis (such as complex state-based arbitration protocols or inter-processor communication task dependencies) which leads to pessimistic over-approximated results compared to state-based methods [17]. Many recent approaches for the software timing analysis on many- and multi-core architectures are built on state-based analysis techniques. The two main considered application classes are streaming applications (modeled as synchronous data flow graphs) [20]–[26] and generic real-time task-based applications [27]–[34]. State-based real-time methods are based on the fact of representing the System Under Analysis (SUA) as a transition system (states and transitions). Since the real operation states of the architecture behaviour are reflected, tighter results can be obtained compared to analytical methods. However, state-based approaches allow exhaustive analysis of system properties at the expense of time-consuming modelling and analysis effort.

In [35], the adoption of model-checking for real-time analysis of SDFGs running on multi-processor with shared communication resources is presented. Especially, it is highlighted in [35] that state-based approaches can hardly address

systems made of high number of heterogeneous components. This approach utilizes timed automata to represent tasks and arbitration protocols for buses and memories. It uses Best- (BCET) and Worst-Case Execution Time (WCET) intervals as lower and upper bounds of the estimated execution time of a specific platform. In our work, we use a timing model based on probability density functions, which allows a more flexible analysis especially in the case of platforms with complex interaction between resources (*e.g.*, cache effect, external shared memory). The approach we investigate uses a combination of probabilistic models and analysis techniques. It is a hybrid analysis approach that can be seen as a trade-off between simulation and state-based verification approaches.

B. Timing analysis approaches based on probabilistic methods

Probabilistic models are frequently adopted to model systems where uncertainty and variability need to be considered. In the context of embedded systems, probabilistic models represent a means of capturing system variability coming from system sensitivity to environment and low level effects of hardware platforms. Probabilistic models (e.g., discrete time Markov chains, Markov automata) can be used to appropriately capture this variability. Probabilistic models are extensions of labeled transition system and allow variations about execution times and state transitions to be considered. Quantitative analysis of probabilistic models can be used to quantify that a given time property is satisfied. Numerical approaches exist that compute the exact measure of the probability at the expense of time-consuming analysis effort. As an illustration, the adoption of probabilistic model checking for evaluation of dynamic data-flow behaviors is presented in [20]. Markov automata is used as the fundamental probabilistic model to capture and analyze architectures. Characteristics as application buffer occupancy, timing performance, and platform energy consumption are estimated. However, this approach is restricted to fully predictable platforms, with low influence of platform resources on timing variations.

Another approach to analyze probabilistic models is to simulate the model for many runs and monitor simulations to approximate the probability that time properties are met. Statistical Model Checking (SMC) has been proposed as an alternative to numerical approaches to avoid an exhaustive exploration of the state-space model. SMC refers to a series of techniques that are used to explore a sub-part of the state-space and provides an estimation about the probability that a given property is satisfied. SMC designates a set of statistical techniques that present the following advantages:

- As classical model checking approach, SMC is based on a formal semantic of systems that allows to reason on behavioral properties. SMC is used to answer qualitative questions (*Is the probability for a model to satisfy a given property greater or equal to a certain threshold?*) and quantitative questions (*What is the probability for a model to satisfy a given property?*).
- It simply requires an executable model of the system that can be simulated and checked against state-based properties expressed in temporal logics. The observed executions are processed to decide with some confidence whether the system satisfies a given property.
- As a simulation-based approach, it is less memory and time intensive than exhaustive approaches.

Various probabilistic model-checkers support statistical model-checking, as for example UPPAAL-SMC [36], Prism [37], and Plasma-Lab [38]. This approach has been considered in various application domains [39].

The usage of UPPAAL-SMC to optimize task allocation and scheduling on multi-processor platform is presented in [40]. Application tasks and processing elements are captured in a Network of Price Timed Automata (NPTA). It is considered that each task execution time follows a Gaussian distribution. In the scope of our work, we especially focus on the preparation process of probabilistic models of execution time. Besides, we consider different platform configurations with different levels of predictability to evaluate the accuracy and analysis time of SMC methods.

Authors in [41], [42] propose a measurement-based approach in combination with hardware and/or software randomization techniques to conduct a probabilistic worst-case execution time (pWCET) through the application of Extreme Value Theory (EVT). In difference to their approach, we apply a Statistical Model Checking (SMC) based analysis capturing the system modus operandi. This enables the obtainement of tighter values compared to the EVT approach. Yet our method could benefit from their measurement methodology.

An iterative probabilistic approach has been presented by Kumar [43] to model the resource contention together with stochastic task execution times to provide estimates for the throughput of SDF applications on multiprocessor systems. Unlike their approach, we apply an SMC based analysis which enables a probabilistic symbolic simulation and the estimation of probabilistic worst-case timing bounds of the target application with estimated confidence values.

In [44] integration of SMC methods in a system-level verification approach is presented. It corresponds to a stochastic extension of the BIP formalism and associated toolset [45]. An SMC engine is presented to sample and control simulation execution in order to decide if the system model satisfies a given property. The preparation process of time annotations in system model is presented in [1] where a statistical inference process is proposed to capture low-level platform effects on application execution. A many-core platform running an image recognition application is considered and stochastic extension of BIP is then used to evaluate the application execution time.

A solution is presented in [46] to apply SMC analysis methods for systems modeled in SystemC. The execution traces of the analyzed model are monitored and a statistical model checker is used to verify temporal properties. The monitor is automatically generated based on a given set of variables to be observed. The statistical model-checker is implemented as a plugin of the Plasma-Lab. In the scope of our work, we adopt the approach presented in [46] to analyze time properties of multi-core systems modeled with SystemC.

The adoption of SMC methods to timing analysis of multi-core systems is promising because it could deliver good compromise between accuracy and analysis time. However, creation of trustful probabilistic models remains a difficult task and is not well supported for multi-core systems. Especially, to the best of our knowledge, no work attempted to systematically evaluate the benefits of using SMC to analyze the timing properties of applications based on Synchronous Data Flow (SDF) model of computation on multi-cores, targeting more tightness of estimated bounds and faster analysis times.

C. Summary of existing analysis approaches

In [47], the authors gave a classification of multi-core architectures w.r.t predictability considerations in their design:

- *Fully timing compositional:* architectures do not exhibit any *timing anomalies*[1] and the usable hardware platform is constraint to be fully compositional. In this case a local worst-case path can be analyzed safely without considering other paths,
- *Compositional with bounded effects:* architectures exhibit timing anomalies but no *domino* effects[2],
- *Non-compositional*: architectures which exhibit both domino effects and timing anomalies. The complexity of timing analysis of such architectures is very high since all paths must be examined due to the fact that a local effect may influence the future execution arbitrarily.

The main discussed approaches are classified in Table I according to the above identified three categories of platforms. We have estimated the efficiency of the approaches for each kind of platforms (well supported ●, partially supported ◓, not well supported ○).

Table I also indicates the position of expected results of our targeted approach. The approach should demonstrate the efficiency of probabilistic methods to analyse multi-core systems. Especially, a significant achievement would be the analysis of non-compositional architectures.

All of the above mentioned analysis approaches rely on the estimation of execution times. For the simulation-based and some probabilistic approaches, the estimation can be performed through a detailed simulation model at instruction- or even cycle-accurate level. This involves very time consuming simulations and belongs to the class of measurement-based execution time analysis techniques, that have to deal with the rare-event problem[3]. Another technique is timing back-annotation to functional or analytical system models, applying timing measurement or static timing analysis approaches. For most real-time systems, the Worst-Case Execution Time (WCET), which is a safe upper bound of all observable execution times, is the most relevant metric. Sometimes also

[1]*Timing anomaly* is referred to "*the situation where a local worst-case doesn't contribute to the global worst-case*" in [47] e.g. shorter execution time of an actor can lead to a larger response time of the application.

[2]*Domino effect* occurs when the execution time difference is arbitrary high (cannot be bounded to a constant) between two states (s, t) of the same program (starting in s, t respectively) on the same hardware [47].

[3]Rare events are events that occur with low frequency but which have potentially widespread impact, e.g. on the execution time.

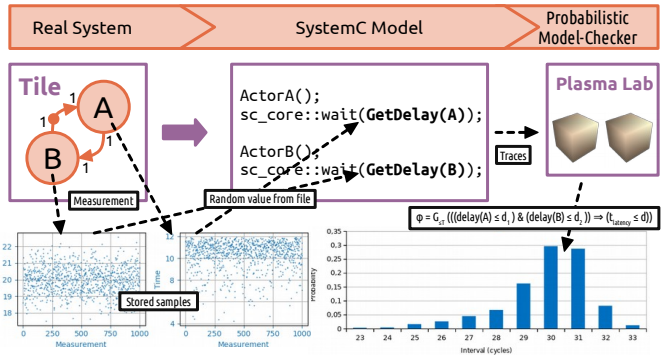


Fig. 1: Example of applying statistical model checking on a two-actor SDF application executed on a single tile. The measured timing characteristic used in a SystemC model can be seen in the left plots. These measured execution time samples were randomly selected by the `GetDelay` function. The blue histogram is the result of the end-to-end latency analysis with Plasma Lab.

the Best-Case Execution Time (BCET) is considered in combination with WCET in a $[\text{BCET}, \text{WCET}]$ interval. Static timing analysis [56] is the state-of-the-art technique to determine the WCET and BCET respectively. With the adoption of todays multi-core systems, limitations of static timing analysis are becoming apparent [47].

For modern multi-core architectures, measurement-based approaches can overcome many problems of static timing analysis. But this comes at a price, since measurement-based timing analysis is facing the rare-event problem, its analysis results are either largely over-approximated (using a large safety margin) or untrustworthy (due to a missing probability analysis). To overcome these challenges for measurement-based timing analysis for modern processors and multi-core systems, the work in the context of the projects PROARTIS[4] [57] and PROXIMA[5] [58] propose a new way for timing analysis. Instead of applying improved static analysis, a measurement based approach in combination with hardware and/or software randomization techniques to conduct a probabilistic worst-case execution time (pWCET) through the application of extreme value theory (EVT) has been proposed. Both projects have successfully assessed that measurement based execution time analysis with the combination of EVT can be used to construct a worst-case probability distribution. In difference to these approaches, we apply an SMC-based analysis approach to consider a full system modus operandi. Furthermore, we are applying the analysis on a set of restricted and well analyzable Models of Computation (MoCs)

The established approach combining probabilistic execution times and probabilistic analysis methods is presented in the next section.

Classes of estimation approaches	Related work	Fully timing compositional	Compositional with bounded effects	Non-compositional
Simulation-based approaches	SystemCoDesigner [8]	◓	○	○
	Daedalus [8]	●	○	○
	SCE [8]	●	○	○
	Koski [7]	◓	○	○
	Kreku et al. [9]	●	○	○
	Arpinen et al. [10]	◓	○	○
Compositional approaches	MPA-RTC [18]	●	◓	○
	SymTA/S [19]	●	◓	○
Holistic approaches	Tendell et al. [48]	●	○	○
	Yen et al. [49]	●	○	○
	Pop et al. [50]	●	◓	○
	Anderson et al. [51]	●	○	○
State-based approaches *SDFGs*	Skelin et al. [21]	●	○	○
	Zhu et al. [22]	●	○	○
	Thakur et al. [23]	●	○	○
	Ahmad et al. [24]	●	○	○
	Malik et. al. [25]	●	○	○
	Yang et al. [26]	●	○	○
	Fakih et. al. [52]	●	◓	○
	Stemmer et. al. [53]	●	◓	○
State-based approaches *Generic tasks*	Norstrom et al. [27]	●	○	○
	Hendrik et al. [28]	●	◓	○
	Lv et al. [29]	●	◓	○
	Gustavsson et al. [30]	●	◓	○
	Giannopoulou et al. [31]	●	◓	○
	Brekling et al. [32]	●	○	○
	Zhang et al. [33]	●	○	○
	Büker et al. [34]	●	◓	○
Probabilistic approaches	BIP-SMC [54]	●	◓	○
	Katoen et al. [20]	●	○	○
	Nouri et al. [1]	●	◓	○
	Stemmer et.al. [55]	●	◓	○
	Expected results from this work	●	◓	◓

TABLE I: Classification of related work and main references.

III. Preliminaries

A. SDFGs

A *synchronous data-flow graph (SDFG)* [59] is a directed graph (see example in Fig.1, Sobel filter in Fig.3a) which consists mainly of nodes (called *actors*) modeling atomic functions/computations and arcs modeling the data flow (called *channels*). SDFGs consume/produce a static number of data samples (*tokens*) each time an actor executes (*fires*). An actor can only be executed, when there are enough tokens available on its ingoing channels.

Definition 1: (SDFG) A synchronous dataflow graph is defined as $SDFG = (\mathcal{A}, \mathcal{C})$, which consist of a finite set $\mathcal{A}$ of actors A and a finite set $\mathcal{C}$ of channels C.

1) a finite set $\mathcal{A}$ of actors A.
2) a finite set $\mathcal{C}$ of channels C. A channel is a tuple $C = (R_i, R_o, B)$ with
   a) The input rate R_i defining the number of tokens that can be written into the channel during the write phase of an actor.
   b) The output rate R_o defining the number of tokens that will be read from the channel during the read phase of an actor.
   c) The buffer size B in number of tokens.

In the context of SDFGs, an *iteration* of an SDFG (refer to [35] for more formal definitions) is completed when the initial tokens' distribution on all its channels is restored. The *end-to-end latency* is the time starting from activating the first instance of the source actor, executing the SDFG application till the last instance of the sink actor is finished.

B. Multi-Core Timing Effects

Variations of the execution time can be caused by two main sources:

1) Software induced: Data dependencies in the algorithm, leading to different execution paths of the software (different branches or number of loop iterations, which are taken).
2) Hardware induced: Depending on the processor and memory architecture, different timing variations can occur, these can be further grouped into
   a) local effects: only dependent on the properties and the state of the processor architecture where the software is executed on. This includes data dependent execution times of instructions, branch prediction, and local cache misses.

[4]Probabilistically Analysable Real-Time Systems (http://www.proartis-project.eu)

[5]Probabilistic real-time control of mixed-criticality multicore and manycore systems (http://www.proxima-project.eu)

b) global effects: dependent on the usage and properties of shared resources used by multiple processors. This includes access to a shared bus, shared memory and shared caches (2nd level cache).

The software induced timing effects can be handled though appropriate algorithm design (i.e. by avoiding data dependent loops) and a static path analysis. Hardware induced local effects can be handled (if known and well documented) by static Worst-Case Execution Time (WCET) analysis. For the global effects, static analysis is extremely difficult or even impossible for specific multi-core platforms. For this reason, measurement based timing analysis approaches are currently being employed to cope with these difficult timing interferences. In this paper we are considering timing effects of all of the described classes.

C. Statistical Model-Checking Methods

Consider a probabilistic system $\mathcal{S}$ and a property φ. Statistical Model Checking (SMC) refers to a series of simulation-based techniques that can be used to answer two types of question [60]: (i) What is the probability that $\mathcal{S}$ satisfies φ (*quantitative analysis*), written $Pr(\mathcal{S} \models \varphi)$? And (ii) Is the probability that $\mathcal{S}$ satisfies φ greater or equal to a threshold θ (*qualitative analysis*), written $Pr_{\geq\theta}(\mathcal{S} \models \varphi)$? The core idea of SMC is to monitor a finite set of simulation traces which are randomly generated by executing the probabilistic system $\mathcal{S}$. Then, statistical algorithms can be used to estimate the probability that the system satisfies a property. In the scope of this paper, we consider two algorithms: *Monte Carlo* with *Chernoff-Hoeffding bound* to answer the quantitative question and *Sequential Probability Ratio Test (SPRT)* for qualitative question. Although SMC only provides an estimation, the algorithms presented below offer strict guarantees on the precision and the confidence of the test.

Let $p = Pr(\mathcal{S} \models \varphi)$ be the probability that the system satisfies a certain property φ. The methods which can quantify p are based on the techniques of Monte Carlo with Chernoff-Hoeffding bound [61]–[63]. Given a precision δ, an estimation procedure consists on computing an approximation p' such that $|p' - p| \leq \delta$ with *confidence* α, i.e., $Pr(|p' - p| \leq \delta) \geq 1 - \alpha$. Given $B_1, ..., B_n$ as n discrete random variables with a Bernoulli distribution of p associated with n simulations of the system. We denote b_i as the outcome of each B_i, which is 1 if the simulation satisfies φ and 0 otherwise. Based on Chernoff-Hoeffding bound [64], p' can be approximated as $p' = (\sum_{i=1}^{n} b_i)/n$ and the minimum number of observations that guarantees $Pr(|p' - p| \leq \delta) \geq 1 - \alpha$ can be computed as $n \geq \frac{4}{\delta^2} \log(\frac{2}{\alpha})$.

The statistical methods to answer the qualitative question are based on *hypothesis testing* (see details in [46], [1]). The principle of these methods is to determine whether the probability that the system satisfies a property is greater than a given bound: $p \geq \theta$? In [65], Younes proposed SPRT as an hypothesis testing algorithm, in which the hypothesis H_0: $p \geq p_0 = \theta + \delta$ is tested against the alternative hypothesis H_1: $p \leq p_1 = \theta - \delta$. The strength of a test is determined by two parameters (α, β) which are the probability of accepting H_0 when H_1 holds (type I error or false negative), and the probability of accepting H_1 when H_0 holds (type II error or false positive). The interval $[p_0, p_1]$ is referred to the *indifference region* of the test which contains θ.

Analysis results using these two statistical algorithms will be presented in Section V.

D. Statistical Timing Properties

In this paper, we consider the properties specified in Bounded Linear Temporal Logic (BLTL). This logic enhances Linear Temporal Logic (LTL) operators with time bounds. A BLTL formula φ is defined over a set of atomic propositions AP, the logical operators (e.g., $true$, $false$, $\neg$, $\rightarrow$ and $\wedge$) and the temporal modal operators (e.g., U, X, F , G, M and W) by the grammar as follows.

$$\varphi := true \mid false \mid ap \in AP \mid \neg\ \varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1\ U_{\leq T}\ \varphi_2$$

The time bounds T is the duration of one simulation run during which we analyze the property. Temporal modality F, for "*eventually*", can be derived from the "*until*" U as $F_{\leq T}\ \varphi$ = $true\ U_{\leq T}\ \varphi$. It means that the property φ is eventually satisfied within T. Similarly, temporal modality G for "*always*" can be derived from F as $G_{\leq T}\ \varphi$ = $\neg F_{\leq T}\ \neg\varphi$. This equation can be explained as: the hypothesis that the property φ is not satisfied within T will not occur or the property φ is always satisfied within T. The semantic of BLTL logic is the semantic of LTL logic restricted to a time interval (see details in [46]). Consider the example in Fig. 1, the computation times of two actors A and B stay below threshold values d_1, d_2: $delay(A) \leq d_1$ and $delay(B) \leq d_2$. The BLTL formula to express the probability that the end-to-end latency $t\_latency$ always stays below a threshold value d within T is $\varphi = G_{\leq T}(((delay(A) \leq d_1)\&(delay(B) \leq d_2)) => (t\_latency \leq d))$.

IV. Approach

Our approach is shown in Fig. 2. We start with a model of our implemented system. This model consists of an SDF computation model of an application that is mapped and scheduled on a multi-processor hardware platform with shared memory communication. The system model is described in section IV-A in detail. The hardware architecture gets instantiated on an FPGA and the modeled application is executed on this system. We then measure the timing behavior of the application running on this system for our model (see arrow ① in Fig. 2). The results of the end-to-end latency analysis of our model is then compared to the actual measured end-to-end latency (see arrow ② in Fig. 2) in section V-B.

A. System Model

In this section, we explain our software and hardware model, as well as the mapping and scheduling of the software on the hardware. Furthermore, we describe how we model the timing behavior of the system.

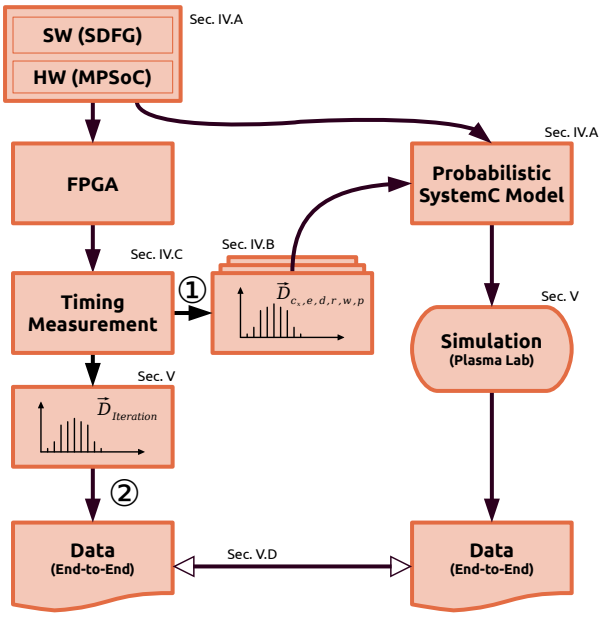


Fig. 2: Overview of our approach to use measured execution time for a simulation-based stochastic end-to-end latency analysis. The annotation refers to the paper section describing the individual steps.

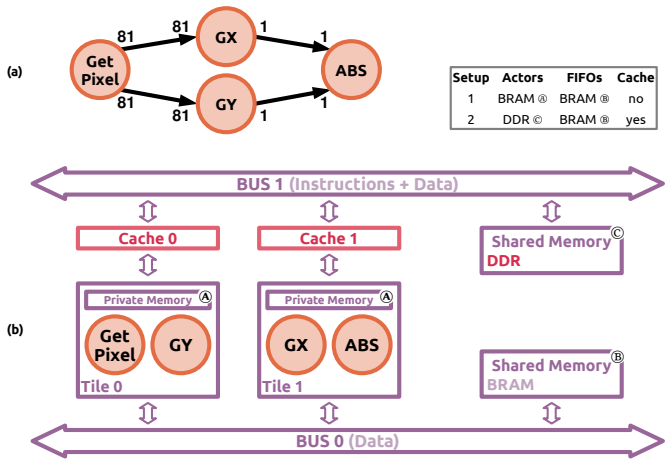


Fig. 3: a) A Sobel filter modeled as an SDFG. A 9×9 pixels segment of an image is read by *GetPixel* actor and is sent to the *GX* and *GY* actors. The result of the convolutions in GX and GY is sent to the *ABS* actor where a pixel for the resulting image is calculated.
b) Our platform consisting out of two tiles with caches and two shared memories. The Sobel filter's actors are mapped to the tiles. The instructions and local data of the actors use either the private memory Ⓐ or the shared DDR Ⓒ. The channels for data exchange between the actors are mapped to the shared BRAM Ⓑ. Due to the added caches and the DDR memory, the timing behavior of the system becomes highly non-deterministic.

1) Model of Computation: The SDF model: The Sobel filter (see Fig. 3a), used in our experiments, follows SDF semantics that are described in subsection III-A.

SDF model of computation offers a strict separation of computation and communication phases of actors. During the computation phase, no interference with any other actor can occur, if the respective code is mapped to dedicated memories (see setup 1 in Fig. 3b). We later weaken this constraint and run the code from a shared DDR memory (see setup 2 in Fig. 3b). This obviously leads to interferences and extra delays between the actors accessing shared memories which will be also taken into consideration by our probabilistic model.

The channels are implemented as FIFO buffers on a shared memory.

The actors are static scheduled and will not be preempted. Iterations can overlap over time. In our example *GetPixel* of one iteration will be executed at the same time *ABS* gets executed from the iteration before (See Fig. 4.

2) Model of Architecture: The hardware model: We started with a hardware model of a composable system that uses multiple independent tiles to execute applications as deterministic as possible. Interference occurs only when applications explicitly access shared resources like shared memories or a shared interconnect. We then extend this hardware with a common memory for the instructions of the software executed on the tiles (Fig. 3b).

Definition 2: (Tile) A tile is a tuple $T = (PE, M_p)$ where PE is the processing element and M_p is the private memory only accessible by the processing element. A tile can execute software without interfering with other tiles as long as the software only accesses the private memory.

Definition 3: (Execution Platform) An execution platform is defined as $EP = (\mathcal{T}, \mathcal{M}_s, \mathcal{C}_a, \mathcal{I})$ consisting of a finite set $\mathcal{T}$ of tiles T as defined in Def. 2, a finite set $\mathcal{M}_s$ of shared memory M_s, a finite set $\mathcal{C}_s$ of dedicated caches C_a for each tile, and a finite set $\mathcal{I}$ of shared interconnects I. While a tile can be connected to multiple interconnects, we assume that every memory is connected to only one interconnect. Furthermore we assume that tiles can only communicate via a shared memory.
In the experiments, without loss of generality, the used interconnects support a first-come first-serve-based communication protocol. The EP_1 and EP_2 for the two setups in Fig. 3b) can be expressed as follows:

$$EP_1 = (\{tile_0, tile_1\}, \{BRAM\}, \{\}, \{BUS_0\}) \tag{1}$$

$$EP_2 = (\{tile_0, tile_1\}, \{DDR, BRAM\}, \{Cache_0, Cache_1\}, \{BUS_0, BUS_1\}) \tag{2}$$

As shown in Fig. 3b, in the first setup (EP_1) the platform consists of two tiles with local data and instruction memories connected with a data bus to a shared memory, through which inter-tile communication takes place. This setup is full composable such that the computation phases of any actor (taking place locally on private memory) can be considered independent from communication phases (taking place via the data bus supporting a single-beat transfer style). For the second setup (EP_2) the computation code of actors is placed on a DDR memory where, with the help of a cache controller (one for each tile), the code is accessed via a data/instruction bus (see BUS1 in Fig. 3b). Also here the inter-tile communication takes place via the data bus (BUS0) and the shared BRAM memory.

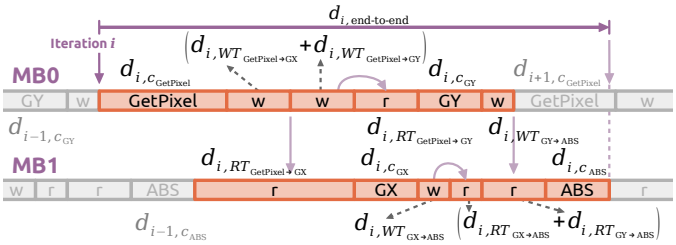


Fig. 4: Expected actor and phase activation for one iteration i (orange - start of the first actor to the end of the last actor) The annotated delays $d_{i,x} \in \vec{D}$ are our measured values for iteration i. $d_{i,RT}$ represent the read-time (equ. 4) and $d_{i,WT}$ the write-time (equ. 3) of the communication. d_{i,c_x} is the computation time an actor x needs in iteration i. The End-To-End latency $d_{i,\text{end-to-end}}$ is the delay of one iteration (cp. [55])

B. Timing Model

All timings are related to the mapped software (i.e. SDF actors and channels) on a specific hardware. For this reason, the annotation of any delays (in cycles) to the models takes place after the mapping and scheduling process, as shown in Fig. 4. While the execution of an actor's computation phase on a specific tile can be represented by a single delay $d \in \vec{D}_c$ (see Def. 4) for each considered execution, the communication timing model requires more effort.

To follow the SDF semantics, communication is realized using a FIFO buffer. Communication between actors is modeled considering every single memory access, including synchronizing meta data, plus some time consumed by the computational parts of the FIFO implementation. For our implementation of the FIFO buffer the time for writing t tokens is described by:

$$d_{WriteTokens}(t) = (n+2)\, d_r + (t+1)\, d_w + n \cdot d_p + d_e \quad (3)$$

The time for reading t tokens to the FIFO is:

$$d_{ReadTokens}(t) = (n+t+2)\, d_r + 2 \cdot d_w + n \cdot d_p + d_d \quad (4)$$

The read and write phase of an actor expects a ready (filled or empty) FIFO buffer for each channel connected to the actor. During these phases the state of the buffer is checked via polling. Each polling iteration takes a time of d_p that is defined by the developer but can vary when executed from shared memory or due to caching effects. The amount n of polling iterations is determined during the analysis of the model and varies between each SDF execution iteration i. Accessing memory consumes read time $d_r \in \vec{D}_r$ or write time $d_w \in \vec{D}_w$ for each token t that is transferred as well as for updating meta data. Furthermore, it takes some computation time for managing the FIFO. These delays are the enqueue delay $d_e \in \vec{D}_e$ of the write phase or the dequeue delay $d_d \in \vec{D}_d$ for the read phase. Delays can vary in each SDF execution iteration i.

Definition 4: (*Time Model*) $\mathcal{D}$ is defined as a set of delay vectors $\vec{D} \in \mathcal{D}$ with probable execution delays, consisting of the following delay vectors:

- $\vec{D}_c \in \mathcal{D}$: The computation time of an actor $A \in \mathcal{A}$.

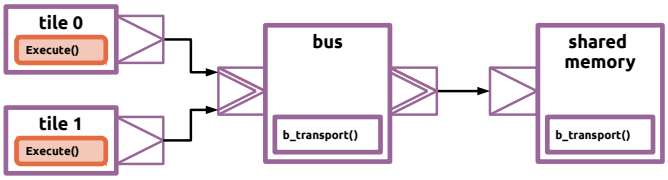


Fig. 5: Setup of our Simulation with SystemC TLM. Two tiles are connected to a shared memory via interconnect. The interconnection to the shared DDR memory (as given in EP_2, see Setup 2 in Fig. 3b) for the instructions and local data is not explicitly modeled in SystemC. They are implicit in the timing data annotated to the model.

- $\vec{D}_e, \vec{D}_d \in \mathcal{D}$: The enqueue and dequeue time that represents the computation time of the FIFO driver.
- $\vec{D}_r, \vec{D}_w \in \mathcal{D}$: The time to read and write to a memory $M \in \mathcal{M}_p \cup \mathcal{M}_s$. For shared memories this delay also includes the overhead of the interconnect $I \in \mathcal{I}$ communication.
- $\vec{D}_p$: A polling delay when an actor $A_x \in \mathcal{A}$ tries to access a FIFO that is blocked (not enough tokens, or not enough free buffer space).

For all above delays, when the corresponding software components need to be loaded from shared memories via caches (like the case in setup 2, see Fig. 3b), the interference with other actors and caching effects is also captured in the corresponding delay vectors. It is assumed that for different mappings the influences of the cache on the execution time of an actor is similar.

C. Probabilistic SystemC Model

The SystemC model is a representation of the whole system. It models the mapped and scheduled SDF application on a specific hardware platform. As shown in Fig. 5 it consists of three major parts: 1) the *tile* modules for simulating the actors' execution on a tile, 2) the *bus* module that simulates the bus behavior (arbitration and transfer) and 3) the *shared memory* module for the read and write access to the shared memory on which the SDF channels are mapped. For this paper, as seen in Fig. 5, we didn't explicitly model the caches, the DDR memory and its bus (BUS1), as given in EP_2 (Setup 2 in Fig. 3b). Instead, for simplification reasons, we capture the timing influence of those components (latency times of the cache, bus transfer and memory latency for loading and storing instructions) in the measured actors' execution times being executed from this memory. It remains interesting for future research to see how the explicit modeling of these elements in the SystemC model would improve the accuracy of EP_2 analysis results.

The behavior of the SDF application is simulated as an *SC_THREAD* of the *tile* module. This module also implements the read and write phases of the actors that run on those tiles. The computation phase of an actor is a simple *wait* statement using the actors' computation time $\vec{D}_c$.

The read and write phase are modeled in more details. They are methods that implement a FIFO behavior that follows the

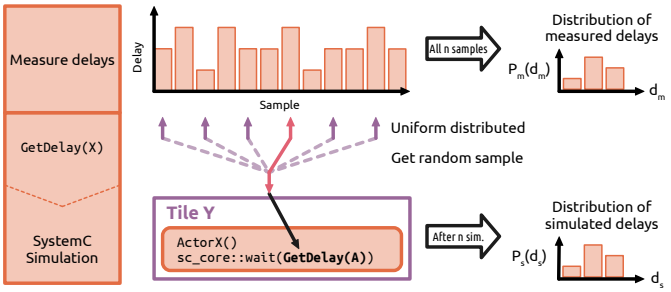


Fig. 6: Process of annotating the SystemC model with measured delays from the real system shown for an example actor *ActorX* on a tile *Y*. The measured delays are stored in a file (top). The SystemC model (bottom) then gets a random value (center) from that file as delay for the simulation. So the distribution of delays for the simulation are similar to the distribution of the measured delays.

SDF semantics. Both, read and write phases are divided into multiple sections. Each section has a different delay annotation $\vec{D_e}$, $\vec{D_d}$ and $\vec{D_p}$ as described in IV-B.

The *bus* module manages the arbitration using *First Come First Server (FCFS)* strategy. The *shared memory* simply distinguishes read ($\vec{D_r}$) and write ($\vec{D_w}$) access. Depending on the access type, it proceeds the simulation time by the related time.

In the SystemC model, the probabilistic timing information can be modeled with the distributions provided by GNU Scientific Library (GSL) [66]. In the case of the Sobel Filter, we consider the computation time of four actors *GetPixel, GX, GY, ABS* and the communication time to read/write data on the channels or on shared memory. In each iteration of the SystemC model, a computation time or communication time is assigned to a value that is either randomly chosen from the *a)* uniform, *b)* normal distributions of the measured delays or *c)* from a text file containing the raw measured delays of software components (we refer to as injected data). Any other distribution can be used.

To apply the uniform distribution, we choose a random value in the interval of [BCET, WCET]. While in the normal distribution, we consider the mean μ and the variance σ of the corresponding measured delays.

To improve the accuracy of the uniform/normal distributions it is desired to model a probability distribution that accurately reflects the distribution of the measured delays. This is realised by reading a random value (see $GetDelay(A)$ in Fig. 6) for every component (e.g. an actor delay) from a text file that provides all measured delays of this component (we refer to as injected data). This process from measuring an actor to simulating its timing behavior is shown in Fig. 6. The same process is applied for modeling the communication. The measured delay of a random iteration of the actual executed actor gets collected in a file containing all samples of the execution times of that specific actor. So after measurement, there exists files of delays for each delay vector described Def. 4 in Sect. IV-B. To use these data inside the simulation, a function `GetDelay` (Fig. 6) selects a random value from the measured raw data file. The distribution of the randomly selected delays is uniform. This achieves a similar delay distribution of the simulated actor compared to the real actor as we will show in the experiments section.

D. Statistical Model-Checking of SystemC Models

The statistical model-checker workflow that we consider takes as inputs a probabilistic model written in SystemC language, a set of observed variables, a BLTL property, and a series of confidence parameters needed by the statistical algorithms, as shown in Fig. 7. First, a monitor model is generated by the *Monitor and aspect-advice generator* (MAG) tool proposed by V.C Ngo in [46]. Then, the generated monitor and the probabilistic system are instrumented and compiled together to build an executable model. In the simulation phase, Plasma Lab iteratively triggers the executable model to run simulations. The generated monitor observes and delivers the execution traces to Plasma Lab. An execution trace contains the observed variables and their simulation instances. The length of traces depends on the satisfaction of the formula to be verified. This length is finite because the temporal operators in the formulas are bounded. Similarly, the required number of execution traces depends on the statistical algorithms in use supported by Plasma Lab (e.g., Monte Carlo with Chernoff-Hoeffding bounds or SPRT).

The executable model is built with three main steps, as illustrated in Fig. 8: *Generation*, *Instrumentation* and *Compilation*. Users create a *configuration file* that especially contains the properties to be verified, the observed variables and the temporal resolution. The configuration file is then used by the MAG tool to generate an aspect-advice file and a monitor model. The aspect-advice file declares the monitor as a *friend* class of the SystemC model, so that the monitor can access to the private variables of the observed model. The monitor model contains a monitor class and a class called $local\_observer$. The $local\_observer$ class contains a callback function that invokes $step()$ function of the appropriate monitor class at a given sampling point during the simulation [67]. The $step()$ function captures the value of the observed variables and their instances to produce execution trace. The temporal resolution specifies when the set of observed variables is evaluated by the monitor during the simulation of the SystemC model. In the *Instrumentation* step, the SystemC model and the generated monitor are instrumented by AspectC++ with the help of the aspect-advice file. The instrumented models are linked to the libraries of SystemC and compiled by *g++ compiler* in the *Compilation* step to build an executable model. Afterward, users run the Plasma Lab to verify the property.

V. Experiments

In this section we describe our experiments and discuss the results.

A. Experiment Definition

In our experiments we use two different configurations of our system. Our platform is shown in Fig. 3b and consists

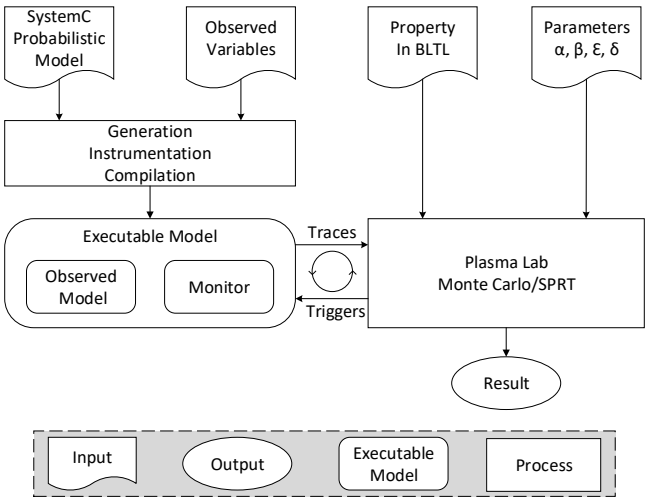


Fig. 7: Generation and instrumentation of SystemC models and interaction with Plasma Lab statistical model checker.

of two MicroBlaze processors as processing elements. First the private memories Ⓐ of the tiles were used for local data (e.g. stack and heap) and instructions of the actors mapped on that tile. Then the private memory gets replaced by a shared memory Ⓒ.

The memory for the local data and instructions are connected to a different interconnect than the shared memory for inter-processor communication. For the communication between the actors, the channels are mapped onto a shared BRAM Memory (Fig. 3b Ⓑ).

In our first experiment, we execute the actors from a private BRAM memory (Fig. 3b Ⓐ) that is exclusively connected to each tile. So there is guaranteed that loading instructions and accessing private data (e.g. from the stack) is deterministic and does not interfere with the other tile.

In the second experiment, we use the shared DDR memory instead of the private memory. (Fig. 3b Ⓒ) The DDR memory is connected via a second bus to the tiles and does not interfere with the bus used to access the SDF channels' FIFO buffers. In this setup, the two tiles are no longer independent from each other. This will lead to a higher variance in the execution time distribution.

The interconnect for inter-tile communication uses First Come First Serve arbitration and a single beat transfer protocol. The caches that are used in the second setup use burst transfer to synchronize with the DDR Memory Ⓒ.

The four actors, *GetPixel*, *GX*, *GY* and *ABS*, of the application (Fig. 3b) are mapped onto the tiles as shown in Fig. 3b. Instructions and local data like the heap and the stack are mapped to private memory Ⓐ or to a shared DDR memory Ⓒ.

In addition, channels used for communication between actors are mapped to the shared Block RAM memory Ⓑ. These channels are organized as FIFO buffers with $4\,\mathrm{Bytes}$ for each token and the buffer size is fixed to the input rate of each actor. The mapping and scheduling is fixed for all experiments.

The measurement technique, we use in the experiments, to get the different delays vectors of different components is based on one presented in [68]. An IP-Core which is connected via a dedicated AXI-Stream bus to communicate with each MicroBlaze processor without interfering with the main system buses, is used. The measured cycle-accurate timings were forwarded via an UART interface without influencing the timing behavior of the platform under observation. In order to achieve that, the code of the application is instrumented in a minimal way (for details refer to [68]).

The multicore system is instantiated on a Xilinx Zynq 7020 with external DDR3-1333 memory.

B. End-to-end Timing Validation

In this subsection, we evaluate the best-case and worst-case end-to-end latencies, denoted in the following as BC latency and WC latency. We declare the end-to-end latency as an observed variable $t\_latency$ in the configuration file used for the SystemC model generation. To quantitatively evaluate the end-to-end latency, the analyzed property is: "*What is the probability that the end-to-end latency stays within an interval [d_1, d_2]?*". This property can be expressed in BLTL with the operators "eventually" as shown in formula 5.

$$\varphi = F_{\leq T}((t\_latency \geq d_1)\&(t\_latency < d_2)) \qquad (5)$$

This property is then analyzed through a finite set of simulations controlled by Plasma Lab. In each simulation, Plasma Lab observes the end-to-end latency of a particular iteration to determine the probability that the end-to-end latency stays in the interval $[d_1, d_2]$. Fig. 9a and Fig. 9b present the probability evolution over the intervals $[d_1, d_2]$ for the two experiments presented in Section V-A. In both experiments, we compare the real measured end-to-end latency data with the analysis results. We use the Monte Carlo algorithm with Chernoff-Hoeffding bound with absolute error $\delta = 0.02$ and confidence $1 - \alpha = 0.98$. It means that the analysis results guarantee the following condition: $Pr(|p' - p| \leq 0.02) \geq 0.98$ (see Section III.C for details).

In the first experiment, the computation time of the actors is modeled by using the uniform and normal distributions. We consider also the injected computation time data. Based on the measured results, the communication time varies in a small interval. Since the measured communication time varies within limited number of values, it is reasonable to apply the uniform distribution to model the variation of the communication time.

Fig. 9a presents the probability evolution over the intervals $[d_1, d_2]$ for the first experiment. The end-to-end latencies obtained from the SystemC model are close to the measured end-to-end latency with a similar range of variation for the different distributions. In the two cases, the WC latencies are over-estimated. We also analyze the SystemC model by applying the injected data which represent the measured data for both computation and communication time. The achieved

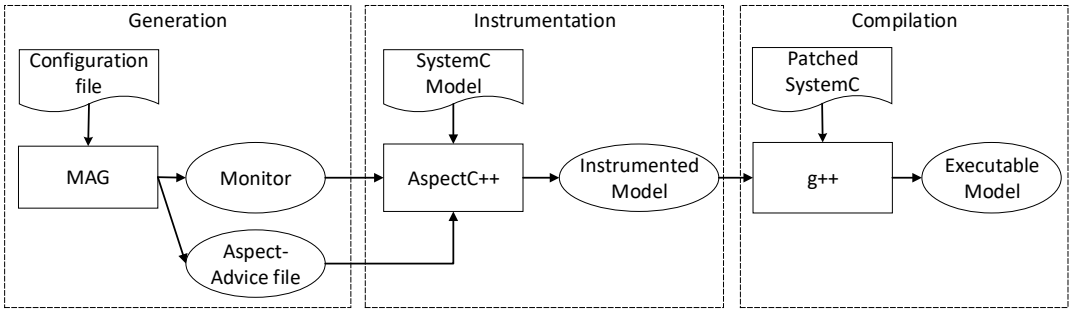


Fig. 8: Successive generation, instrumentation and compilation steps to build SystemC executable model.

TABLE II: Comparison of the end-to-end latencies of the analysis with the different distributions compared with the measured data. While for setup I a normal distribution is sufficient, in setup II the observed worst-case was only covered with the actual distribution of the delays.

Subject	**BC latency I**	**WC latency I**	**BC latency II**	**WC latency II**
Measured data	38170	38806	290004	293970
Uniform/Uniform	[39300, 39400]	[40000, 40100]	[277000, 278000]	[292000, 293000]
Normal/Uniform	[39400, 39500]	[39900, 40000]	[277000, 278000]	[291000, 292000]
Injected data	[39400, 39500]	[39900, 40000]	[319000, 320000]	[327000, 328000]

results do not significantly variate between the uniform, normal distributions and the injected data.

In the second experiment, we follow the same analysis process, as presented in the first experiment. In Fig. 9b, the uniform and normal distributions clearly under-estimate the WC latency. Moreover, the interval of variation is larger than the interval of the measured latency. However, the achieved end-to-end latency of the injected data clearly over-approximates the measured WC latency since the chosen delay values are based on the exact measured delay values. The interval of variation is also closer to the interval of the measured latency. This can be explained that the caches in this experiment cause a large variation in communication time. The applied uniform distribution shows a poor representation of the variation of the communication time in such a complex architecture.

The bias between the real-measured latency and the simulated latencies comes from a pessimistic communication model and the lack of consideration of data dependencies between actors. In the real application, the two actors *GX* and *GY* have equal execution time in one iteration due to there symmetry. The *ABS* actor waits for the slowest dependent actor (*GX*, *GY*). In our simulation the execution times of *GX* and *GY* gets selected randomly and independently. So that the measured timings for faster executions get covered by the slower execution times.

Tab. II summarizes the BC latency and WC latency obtained from the analysis with the uniform, normal distribution and the injected data compared with the measured data in the two experiments. Given an interval, it takes from 10 minutes to 40 minutes to analyze the property. Further analysis to identify precisely the worst-case end-to-end latency is presented in the next section.

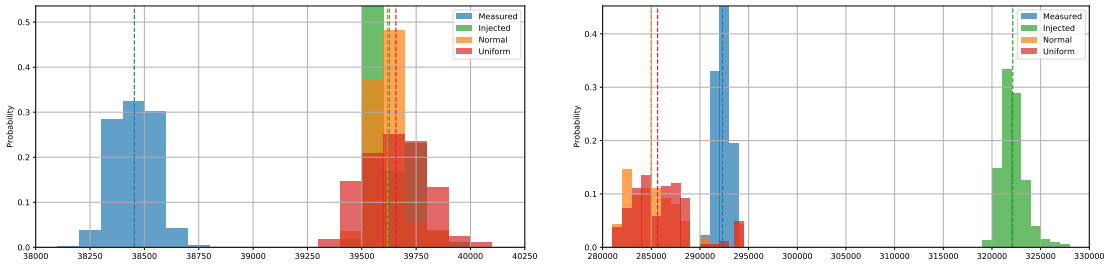
C. Evaluation of Statistical Model Checking Methods

We want now to bound the end-to-end latency within a threshold value d for the two experiments. Thus, we analyze the property: *"The cumulative probability that the end-to-end latency ($t\_latency$) stays below time bound d"*. This property can be translated in BLTL with the operator "always" as follows (Formula 6).

$$\varphi = G_{\leq T}(t\_latency \leq d) \tag{6}$$

The temporal modal operator G is applied that allows us to check whether all the end-to-end latencies of several successive iterations during the simulation time T stay below d. Therefore, the worst-case end-to-end latency obtained from this property is much more accurate than in the first property. Fig. 10 presents the cumulative probability of the end-to-end latency over time bound d for the first experiment. We compare the analyzed results using the uniform distribution, normal distribution and the injected computation time data. The cumulative probability converges to 1 which means that all the end-to-end latencies analyzed stay below the value d. We can consequently bound the WC latency of the normal distribution and injected computation time under $40\,000\,\mathrm{cycles}$ ($3.08\,\%$ higher than the measured WC latency). The bound of WC latency of the uniform distribution is $40\,100\,\mathrm{cycles}$ ($3.33\,\%$ higher than the measured WC latency).

In the second experiment, the same evolution of the cumulative probability is observed. However, the results obtained from the normal and uniform distributions under-estimate the WC latency. Only the injected data can provide an over-approximation of the WC latency. Thus, we present the cumulative probabilities of the SystemC model with normal distribution and the injected data, as shown in Fig. 11. The injected data can provide an over-approximation of the WC latency. The bound of WC latency of the injected data is $328\,000\,\mathrm{cycles}$



(a) without caches **(b)** with caches

Fig. 9: Histogram of end-to-end latency analysis with normal distribution for computation time and different distributions for communication time. Uniform and Normal distributions as well as the injected delay data are compared with the real measured end-to-end latency (measured data).

(11.57 % higher than the measured WC latency). While the normal distribution bounds the WC latency of 324 900 cycles (10.52 % higher than the measured WC latency). For each threshold value d, it takes from several minutes to one hour to analyze.

In the following, we analyze timing properties of the SystemC model using the algorithm SPRT. SPRT checks if the probability to satisfy a property is above or below to a given bound θ. In Plasma Lab, SPRT first estimates a probability to satisfy a property and then this estimated probability is compared with θ to answer the qualitative question. We apply SPRT to analyze the second property with the parameters: $\alpha = \beta = 0.001$, $\delta = 0.01$ and a bound θ.

In the first experiment, we compare the results obtained by Monte Carlo with Chernoff-Hoeffding bound and SPRT applying the normal/uniform distributions for computation/communication time. As illustrated in Fig.12, SPRT shows an over approximation on the probability compared to Monte Carlo with Chernoff-Hoeffding bound. The WC latency obtained by SPRT can be bounded with a closer value of time bound d to the measured WC latency (39 950 cycles compared to 40 000 cycles as in Monte Carlo with Chernoff-Hoeffding bound). Moreover, the analysis with SPRT costs significantly less time than Monte Carlo with Chernoff-Hoeffding bound. In one analysis, SPRT takes 4 minutes compared to 20 minutes in Monte Carlo with Chernoff-Hoeffding bound.

In the second experiment, we compare the results obtained by Monte Carlo with Chernoff-Hoeffding bound and SPRT applying the injected data. In Fig. 13, SPRT leads to under or over approximate the probability. The WC latency is bounded under 327 000 cycles compared to 328 000 cycles as in Monte Carlo with Chernoff-Hoeffding bound. The analysis time is 1 min for one analysis compared to 40 minutes in Monte Carlo with Chernoff-Hoeffding bound. Tab. III summarizes the results of the over-approximation and the analysis time of the two experiments. The overall analysis time of two algorithms Monte Carlo with Chernoff-Hoeffding bound and SPRT depends on the number of simulations and the analysis time of each simulation. In the case of Monte Carlo with Chernoff-Hoeffding bound, the number of simulations is constant in both

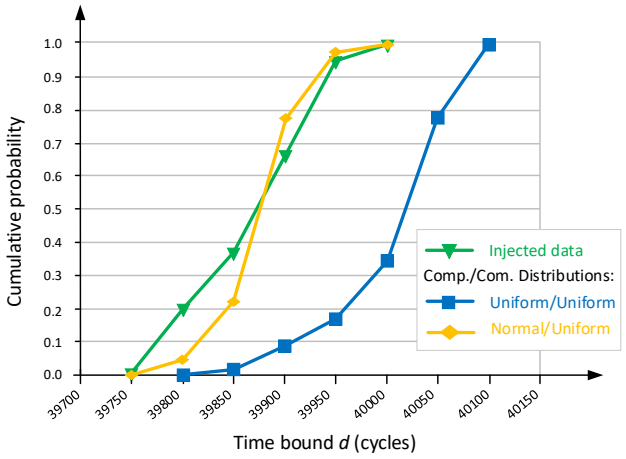


Fig. 10: Cumulative probability that the end-to-end latency is less or equal to a threshold value d, analyzed by Monte Carlo with Chernoff-Hoeffding bound algorithm in the first experiment.

two configurations because of the same values of the precision δ and the confidence σ. Therefore, the higher complexity of system in the second configuration makes a higher overall analysis time. However, in the case of SPRT, the number of simulations depends on the satisfaction of the ratio test (see details in [1]) and the higher complexity of system in the second configuration only makes a higher analysis time of each simulation. The smaller number of simulations analyzed in the second configuration causes the less overall analysis time compared to the first configuration.

Most of the time, it is not necessary to identify precisely the probability to satisfy a property and we only want to bound this probability with a threshold value θ. In these cases, SPRT is more efficient than Monte Carlo with Chernoff-Hoeffding bound in terms of analysis time.

VI. Summary & Future Work

In this work, we have presented a framework that is used to evaluate the efficiency of SMC methods to analyse real-time properties of SDF applications running on multicores.

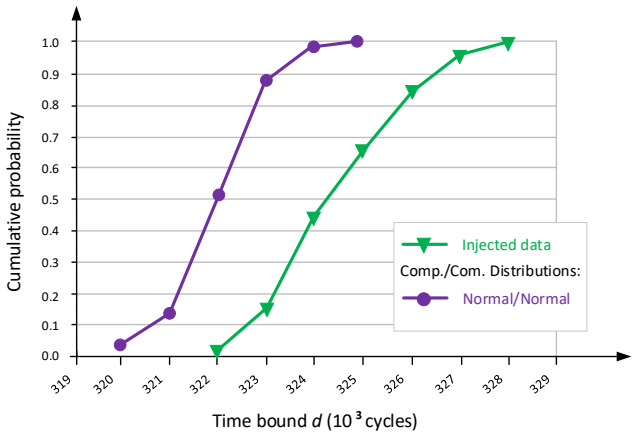


Fig. 11: Cumulative probability that the end-to-end latency is less or equal to a threshold value d, analyzed by Monte Carlo with Chernoff-Hoeffding bound algorithm in the second experiment.

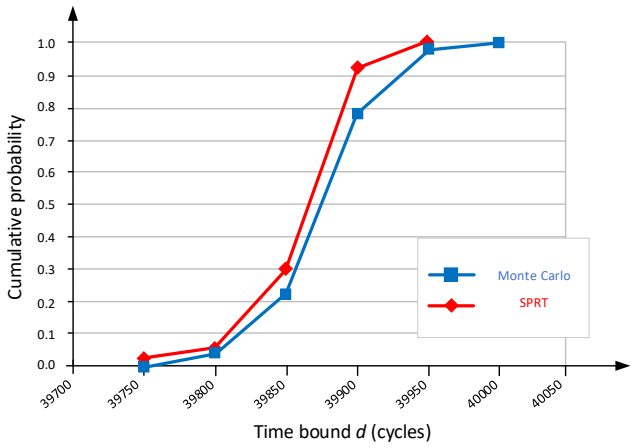


Fig. 12: Cumulative probability that the end-to-end latency stays below a threshold value d, analyzed by Monte Carlo with Chernoff-Hoeffding bound and SPRT in the first experiment.

Our approach uses real measured execution times to annotate a probabilistic SystemC model. The viability of our approach was demonstrated on a Sobel filter running on a 2 tiles platform implemented on top of a Xilinx Zynq 7020. In contrast to traditional real-time analysis methods the SMC approach requires a more sophisticated model of the execution time distribution and thus can tackle the limitations to deliver fast yet accurate timing estimations. Our experiments showed that the selection of the probabilistic distribution function is crucial for the quality of analysis results. In the two considered experiment, worst case end-to-end latency was estimated with an approximation close to 3 % and 11 % compared to real implementations. The SPRT simulation method proved significantly reduced analysis time compared to Monte-Carlo. In future work we plan to use the Kernel Density Estimation technique [69] that will be applied to estimation a more accurate PDF from measured data.

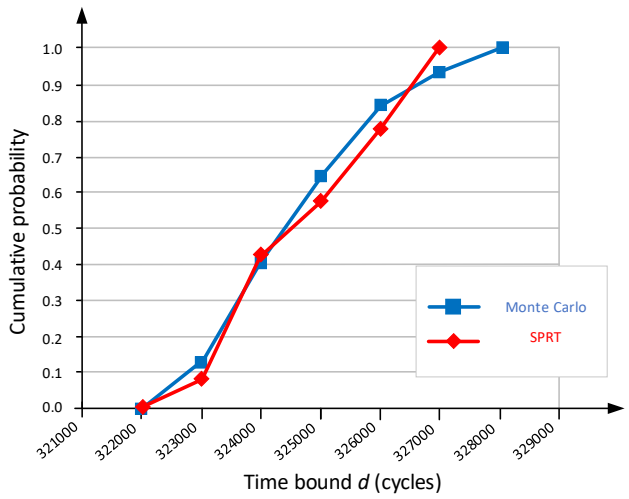


Fig. 13: Cumulative probability that the end-to-end latency stays below a threshold value d, analyzed by Monte Carlo with Chernoff-Hoeffding bound and SPRT in the second experiment.

TABLE III: Estimation the WC latency with the analysis accuracy and duration.

Subject	**WC latency I**	**WC latency II**
Measured data	38806	293970
Over-approximation		
Monte Carlo	3.08%	11.58%
SPRT	2.94%	11.23%
Analysis time		
Monte Carlo	20 mins	40 mins
SPRT	4 mins	1 min

REFERENCES

[1] A. Nouri, M. Bozga, A. Moinos, A. Legay, and S. Bensalem, "Building faithful high-level models and performance evaluation of manycore embedded systems," in *ACM/IEEE International conference on Formal methods and models for codesign*, 2014.

[2] A. Legay, B. Delahaye, and S. Bensalem, "Statistical model checking: An overview," in *Runtime Verification*, H. Barringer, Y. Falcone, B. Finkbeiner, K. Havelund, I. Lee, G. Pace, G. Roşu, O. Sokolsky, and N. Tillmann, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 122–135.

[3] I. S. Association *et al.*, "Ieee standard for standard systemc language reference manual," *IEEE Computer Society*, 2012.

[4] J. Keinert, M. Streubühr, T. Schlichter, J. Falk, J. Gladigau, C. Haubelt, J. Teich, and M. Meredith, "Systemcodesigner-an automatic esl synthesis approach by design space exploration and behavior synthesis for streaming applications," *ACM Trans. Des. Autom. Electro. Syst.*, vol. 14, no. 1, pp. pp.1–23, 1 2009.

[5] C. Erbas, A. D. Pimentel, M. Thompson, and S. Polstra, "A framework for system-level modeling and simulation of embedded systems architectures," *EURASIP Journal on Embedded Systems*, 2007.

[6] R. Dmer, A. Gerstlauer, J. Peng, D. Shin, L. Cai, H. Yu, S. Abdi, and D. Gajski, "System-on-chip environment: A specc-based framework for heterogeneous mpsoc design," *EURASIP Journal on Embedded Systems*, vol. 2008, 2008.

[7] T. Kangas, P. Kukkala, H. Orsila, E. Salminen, and H. T.D., "Uml-based multiprocessor soc design framework," *ACM Transactions on Embedded Computing Systems*, vol. 5, no. 2, pp. 281–320, 5 2006.

[8] A. Gerstlauer, C. Haubelt, A. D. Pimentel, T. Stefanov, D. D. Gajski, and J. Teich, "Electronic system-level synthesis methodologies," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1517–1530, 2009.

[9] J. Kreku, M. Hoppari, T. Kestilä, Y. Qu, J. Soininen, P. Andersson, and K. Tiensyrjä, "Combining uml2 application and systemc platform modelling for performance evaluation of real-time embedded systems," *EURASIP Journal on Embedded Systems*, vol. 2008, pp. 6:1–6:18, 1 2008.

[10] T. Arpinen, E. Salminen, T. D. Hämäläinen, and M. Hännikäinen, "Performance evaluation of uml-2 modeled embedded streaming applications with system-level simulation," *EURASIP Journal on Embedded Systems*, vol. 2009, 2009.

[11] Intel, "Intel cofluent studio," http://www.intel.com/content/www/us/en/cofluent/intel-cofluent-studio.html.

[12] T. Architect, http://www.timing-architects.com.

[13] ChronSIM, http://www.inchron.com/tool-suite/chronsim.html.

[14] TraceAnalyzer, http://www.symtavision.com/products/symtastraceanalyzer/.

[15] SpaceCoDesign, www.spacecodesign.com.

[16] MirabilisDesign, www.mirabilisdesign.com.

[17] S. Perathoner, E. Wandeler, L. Thiele, A. Hamann, S. Schliecker, R. Henia, R. Racu, R. Ernst, and M. González Harbour, "Influence of different abstractions on the performance analysis of distributed hard real-time systems," *Des. Autom. Embedded Syst.*, vol. 13, no. 1-2, pp. 27–49, Jun. 2009. [Online]. Available: http://dx.doi.org/10.1007/s10617-008-9015-1

[18] 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.*, vol. 11, no. 1, pp. 8:1–8:23, 4 2012. [Online]. Available: http://doi.acm.org/10.1145/2146417.2146425

[19] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System Level Performance Analysis - the SymTA/S Approach," in *IEE Proceedings Computers and Digital Techniques*, 2005.

[20] J.-P. Katoen and H. Wu, "Probabilistic model checking for uncertain scenario-aware data flow," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 22, no. 1, pp. 15:1–15:27, Sep. 2016. [Online]. Available: http://doi.acm.org/10.1145/2914788

[21] M. Skelin, E. R. Wognsen, M. C. Olesen, R. R. Hansen, and K. G. Larsen, "Model checking of finite-state machine-based scenario-aware dataflow using timed automata," in *Industrial Embedded Systems (SIES), 2015 10th IEEE International Symposium on*. IEEE, 2015, pp. 1–10.

[22] X.-Y. Zhu, R. Yan, Y.-L. Gu, J. Zhang, W. Zhang, and G. Zhang, "Static Optimal Scheduling for Synchronous Data Flow Graphs with Model Checking," in *FM 2015: Formal Methods*. Springer, 2015, pp. 551–569. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-319-19249-9_34

[23] R. K. Thakur and Y. N. Srikant, "Efficient Compilation of Stream Programs for Heterogeneous Architectures: A Model-Checking based approach," Indian Institute of Science, India, Tech. Rep. IISc-CSA-TR-2015-2, 2015. [Online]. Available: http://www.csa.iisc.ernet.in/TR/2015/2/TechReport2015.pdf

[24] W. Ahmad, E. de Groote, P. K. Hlzenspies, M. I. A. Stoelinga, and J. C. van de Pol, "Resource-constrained optimal scheduling of synchronous dataflow graphs via timed automata," in *Proceedings of 14th IEEE International Conference on Application of Concurrency to System Design (ACSD)*. IEEE, 2014.

[25] A. Malik and D. Gregg, "Orchestrating Stream Graphs Using Model Checking," *ACM Trans. Archit. Code Optim.*, vol. 10, no. 3, pp. 19:1–19:25, 9 2013. [Online]. Available: http://doi.acm.org/10.1145/2512435

[26] Y. Yang, M. Geilen, T. Basten, S. Stuijk, and H. Corporaal, "Automated bottleneck-driven design-space exploration of media processing systems," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '10. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2010, pp. 1041–1046.

[27] C. Norstrom, A. Wall, and W. Yi, "Timed automata as task models for event-driven systems," in *Real-Time Computing Systems and Applications, 1999. RTCSA'99. Sixth International Conference*. IEEE, 1999, pp. 182–189.

[28] M. Hendriks and M. Verhoef, "Timed automata based analysis of embedded system architectures," in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*. IEEE, 2006, pp. 8–pp.

[29] M. Lv, W. Yi, N. Guan, and G. Yu, "Combining Abstract Interpretation with Model Checking for Timing Analysis of Multicore Software," in *2010 31st IEEE Real-Time Systems Symposium*, 2010, pp. 339–349.

[30] A. Gustavsson, A. Ermedahl, B. Lisper, and P. Pettersson, "Towards WCET analysis of multicore architectures using UPPAAL," in *WCET*, 2010, pp. 101–112.

[31] G. Giannopoulou, K. Lampka, N. Stoimenov, and L. Thiele, "Timed model checking with abstractions: Towards worst-case response time analysis in resource-sharing manycore systems," in *Proc. International Conference on Embedded Software (EMSOFT)*. Tampere, Finland: ACM, 10 2012, pp. 63–72.

[32] A. Brekling, M. R. Hansen, and J. Madsen, "Models and formal verification of multiprocessor system-on-chips," *The Journal of Logic and Algebraic Programming*, vol. 77, no. 1-2, pp. 1–19, 9 2008.

[33] W. Zhang, "Bounding Worst-Case Performance for Multi-Core Processors with Shared L2 Instruction Caches," *Journal of Computing Science and Engineering*, vol. 5, no. 1, pp. 1–18, 2011.

[34] M. Bker, "An Automated Semantic-Based Approach for Creating Task Structures," Dissertation, University of Oldenburg, 2013.

[35] M. Fakih, K. Grüttner, M. Fränzle, and A. Rettberg, "State-based real-time analysis of SDF applications on mpsocs with shared communication resources," *Journal of Systems Architecture - Embedded Systems Design*, vol. 61, no. 9, pp. 486–509, 2015.

[36] A. David, K. G. Larsen, A. Legay, M. Mikucionis, D. B. Poulsen, J. van Vliet, and Z. Wang, "Statistical model checking for networks of priced timed automata," in *Formal Modeling and Analysis of Timed Systems*, ser. Lecture Notes in Computer Science, U. Fahrenberg and S. Tripakis, Eds. Springer Berlin Heidelberg, 2011, vol. 6919, pp. 80–96.

[37] M. Kwiatkowska, G. Norman, and D. Parker, "Prism 4.0: Verification of probabilistic real-time systems," in *In Proc. International Conference on Computer Aided Verification (CAV'11)*, 7 2011, pp. 585–591.

[38] C. Jegourel, A. Legay, and S. Sedwards, "A platform for high performance statistical model checking - plasma," in *In Proc. International Conference Tools and Algorithms for the Construction and Analysis of Systems (TACAS'12)*, 2012, pp. pp.498 – 503.

[39] P. Bulychev, A. David, K. Larsen, M. Mikucionis, D. B. Poulsen, A. Legay, and Z. Wang, "Statistical model checking for priced timed automata," in *In Proc. 10th workshop on quantitative aspects of programming languages and systems (QAPL'12)*, 2012.

[40] M. Chen, D. Yue, X. Qin, X. Fu, and P. Mishra, "Variation-aware evaluation of mpsoc task allocation and scheduling strategies using statistical model checking," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, pp. 199–204.

[41] F. J. Cazorla, E. Quiñones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston *et al.*, "Proartis: Probabilistically analyzable real-time systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, no. 2s, p. 94, 2013.

[42] F. J. Cazorla, J. Abella, J. Andersson, T. Vardanega, F. Vatrinet, I. Bate, I. Broster, M. Azkarate-Askasua, F. Wartel, L. Cucu *et al.*, "Proxima: Improving measurement-based timing analysis through randomisation and probabilistic analysis," in *Digital System Design (DSD), 2016 Euromicro Conference on*. IEEE, 2016, pp. 276–285.

[43] A. Kumar, "Analysis, design and management of multimedia multiprocessor systems," Ph.D. dissertation, Eindhoven University of Technology, 2009.

[44] A. Nouri, S. Bensalem, M. Bozga, B. Delahaye, C. Jegourel, and A. Legay, "Statistical model checking qos properties of systems with sbip," *International Journal on Software Tools for Technology Transfer*, vol. 17, no. 2, pp. 171–185, 2014.

[45] A. Basu, B. Bensalem, M. Bozga, J. Combaz, M. Jaber, T. Nguyen, and J. Sifakis, "Rigorous component-based system design using the bip framework," *IEEE Software*, vol. 28, no. 3, pp. 41–48, May 2011.

[46] V. C. Ngo, A. Legay, and J. Quilbeuf, "Statistical model checking for systemc models," *2016 IEEE 17th International Symposium on High Assurance Systems Engineering*, pp. 197–204, 2016.

[47] C. Cullmann, C. Ferdinand, G. Gebhard, D. Grund, C. Maiza, J. Reineke, B. Triquet, and R. Wilhelm, "Predictability considerations in the design of multi-core embedded systems," in *Proceedings of the Embedded Real Time Software and Systems Congress (ERTS^2) 2010*, 2010.

[48] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocess. Microprogram.*, vol. 40, no. 2-3, pp. 117–134, 4 1994. [Online]. Available: http://dx.doi.org/10.1016/0165-6074(94)90080-9

[49] T.-Y. Yen and W. Wolf, "Performance estimation for real-time distributed embedded systems," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 9, no. 11, pp. 1125–1136, 11 1998.

[50] T. Pop, P. Eles, and Z. Peng, "Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems," in *Proceedings of the tenth international symposium on Hardware/software codesign*. ACM, 2002, pp. 187–192. [Online]. Available: http://dl.acm.org/citation.cfm?id=774828

[51] B. Andersson, A. Easwaran, and J. Lee, "Finding an upper bound on the increase in execution time due to contention on the memory bus in COTS-based multicore systems," *ACM Sigbed Review*, vol. 7, no. 1, p. 4, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1851172

[52] M. Fakih, K. Grttner, M. Frnzle, and A. Rettberg, "State-based real-time analysis of SDF applications on MPSoCs with shared communication resources," *Journal of Systems Architecture*, vol. 61, no. 9, pp. 486 – 509, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762115000326

[53] R. Stemmer, M. Fakih, K. Grttner, and W. Nebel, "Towards state-based RT analysis of FSM-SADFGs on MPSoCs with shared memory communication," in *9th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO) 2017*, 1 2017.

[54] A. Nouri, S. Bensalem, M. Bozga, B. Delahaye, C. Jegourel, and A. Legay, "Statistical model checking qos properties of systems with sbip," *International Journal on Software Tools for Technology Transfer*, p. pp. 14, 2014.

[55] R. Stemmer, H. Schlender, M. Fakih, K. Grttner, and W. Nebel, "Probabilistic state-based RT-analysis of SDFGs on MPSoCs with shared memory communication," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 3 2019.

[56] R. Wilhelm, S. Altmeyer, C. Burguière, D. Grund, J. Herter, J. Reineke, B. Wachter, and S. Wilhelm, *Static Timing Analysis for Hard Real-Time Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 3–22. [Online]. Available: https://doi.org/10.1007/978-3-642-11319-2_3

[57] F. J. Cazorla, E. Quiñones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim, "Proartis: Probabilistically analyzable real-time systems," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 2s, pp. 94:1–94:26, May 2013. [Online]. Available: http://doi.acm.org/10.1145/2465787.2465796

[58] F. J. Cazorla, J. Abella, J. Andersson, T. Vardanega, F. Vatrinet, I. Bate, I. Broster, M. Azkarate-askasua, F. Wartel, L. Cucu, F. Cros, G. Farrall, A. Gogonel, A. Gianarro, B. Triquet, C. Hernández, C. Lo, C. Maxim, D. Morales, E. Quiñones, E. Mezzetti, L. Kosmidis, I. Agirre, M. Fernández, M. Slijepcevic, P. Conmy, and W. Talaboulma, "PROXIMA: improving measurement-based timing analysis through randomisation and probabilistic analysis," in *DSD*. IEEE Computer Society, 2016, pp. 276–285.

[59] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[60] A. Legay, B. Delahaye, and S. Bensalem, "Statistical model checking: An overview," *International conference on runtime verification*, pp. 122–135, 2010.

[61] C. P. Robert, G. Casella, and G. Casella, *Introducing monte carlo methods with r*. Springer, 2010, vol. 18.

[62] T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet, *Approximate probabilistic model checking*. Springer, 2004, no. 73–84.

[63] R. Grosu and S. A. Smolka, *Monte carlo model checking*. Springer, 2005, no. 271–286.

[64] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 13–30, 1963.

[65] H. L. Younes, "Verification and planning for stochastic processes with asynchronous events," Ph.D. dissertation, Carnegie Mellon University, 2005.

[66] GSL, "https://www.gnu.org/software/gsl/."

[67] S. Dutta, D. Tabakov, and M. Y. Vardi, "Chimp: a tool for assertion-based dynamic verification of systemc models," *Program Proceedings*, p. 38, 2013.

[68] C. Schlaak, M. Fakih, and R. Stemmer, "Power and execution time measurement methodology for sdf applications on fpga-based mpsocs," *arXiv preprint arXiv:1701.03709*, 2017.

[69] E. Parzen, "On estimation of a probability density function and mode," *Ann. Math. Statist.*, vol. 33, no. 3, pp. 1065–1076, 09 1962. [Online]. Available: https://doi.org/10.1214/aoms/1177704472