Experimental Evaluation of Statistical Model Checking Methods for Probabilistic Timing Analysis of Multiprocessor Systems

Hai-Dang Vu, Sébastien Le Nours, Sébastien Pillement
University of Nantes, IETR UMR CNRS 6164, France
{hai-dang.vu, sebastien.le-nours, sebastien.pillement}@univ-nantes.fr

***Abstract*—Timing prediction of complex parallel data flow applications on multiprocessor systems represents a difficult task due to complex interferences caused by platforms shared resources. In this domain, classical analytical or simulation-based approaches demonstrate scalability issues to deliver fast yet accurate predictions. In this work, we present an experimental evaluation of new simulation-based statistical methods for timing analysis of multiprocessor systems. We adopt a measurement-based approach for the creation of probabilistic system-level models of the studied systems. Efficiency of statistical methods is evaluated for platforms with different levels of complexity from the point of view of shared resources. We compare our approach against measurement and traditional simulation methods on two case-studies from the computer vision domain: a Sobel filter and a JPEG decoder. We show that our simulation approach has good potential for fast yet accurate design space exploration.**

***Index Terms*—Simulation-based performance estimation, Multi Processor, Probabilistic timing analysis**

I. Introduction

Because of the growing demand in computational efficiency, more and more embedded systems are designed around multiprocessor platforms. Validation of software running on such platforms is fundamental to guarantee functional correctness and that timing requirements are fully met. However, software performance prediction on multiprocessor platforms is becoming more complex due to significant influence of shared resources (*e.g.*, communication buses, memories). Intensive analysis of multiprocessor systems under potential working scenarios is thus mandatory to prevent costly design cycles.

In this context, probabilistic models are a useful solution to capture execution time variation of application software running on a multiprocessor platform. Statistical analysis methods can be used to quantify the probability that a given temporal requirement is satisfied and deliver thus useful results early in the design process. Many algorithms are based on model checking techniques to compute the probability by a numerical approach [1]. Another approach to evaluate probabilistic models, called statistical model checking (SMC) [2], is to simulate the created probabilistic model for many runs and monitor simulations to approximate the probability that requirements are met. The adoption of SMC techniques for analysis of probabilistic models represents a far less memory and time intensive approach than exhaustive formal methods. However, creation of trustful probabilistic models of multiprocessor systems represents a significant effort and SMC methods have still rarely been adopted to analyze temporal properties of applications mapped on such platforms with complex hierarchy of shared resources.

In this work, we present a systematic evaluation of SMC methods for timing analysis of multiprocessor systems. A modeling and characterization workflow is presented to set-up system-level models annotated with timings inferred from measurements on a real prototype. In our approach, applications are described as synchronous data flow graphs (SDFGs) mapped on a tile-based platform with different levels of complexity in terms of shared resources. Probabilistic models are used to capture variations in application computation times caused by data-dependent software execution paths. Probabilistic models are also used to model influences of platform's shared resources on computation times and communication times between application's actors. SMC simulation methods are then considered to control simulation runs and deliver quantitative and qualitative answers to temporal requirements. Achieved level of accuracy and simulation runtime are evaluated through different use-cases and platform configurations.

The remainder of this paper is as follows. In Section II, we present and compare existing performance evaluation approaches of multiprocessor systems that use SMC methods. In Section III, we provide an overview of the main concepts associated with our approach, while Section IV presents our modeling approach and the way created probabilistic models are adapted to allow simulation with SMC methods. In Section V, we present our use-cases, the experimental setup, our comparison results and a discussion of these results. The paper closes with a summary and an outlook on future work.

II. Related Work

The adoption of SMC methods for performance estimation of multiprocessor systems was notably considered by Nouri et al. in [3], [4]. The proposed framework corresponds to an extension of the BIP formal component-based framework with stochastic models [5]. In [3], a measurement-based approach was presented to favor creation of system-level models annotated with timings following distribution functions (*e.g.*, normal, exponential). A many-core platform running an image recognition application was considered as a use-case to evaluate accuracy and analysis time of the approach.

The usage of UPPAAL-SMC [6] to optimize tasks allocation and scheduling on multiprocessor platform was presented in [7]. Application tasks and processing elements were captured in a network of priced timed automata with execution time following a Gaussian distribution. In the scope of our work, we further focus on the preparation process of probabilistic models and their analysis with SMC methods. In [8], the adoption of UPPAAL-SMC for real-time analysis of SDFGs running on a multiprocessor platform with shared communication resources is presented. The evaluation of the approach was led with a Sobel filter running on top of a two processors platform. This work illustrated the modeling effort required to get reliable descriptions of tasks and arbitration protocols for shared resources. In this work, we adopt similar assumptions as [8] about the system model. Compared to [4], [7], [8], we focus on application of SMC methods to SystemC models as it allows adding functional behaviors to models. A solution was presented in [9] to apply SMC methods for systems modeled in SystemC. A monitoring approach was presented to allow execution traces to be analyzed by the statistical model checker Plasma-lab [10]. This approach was evaluated with a simple embedded control system. In [11], a first evaluation of the efficiency of SMC methods for SystemC models of multiprocessor systems was presented but on a platform with limited complexity. In the scope of this paper, we evaluate the efficiency of SMC methods for SystemC models through different use-cases with different levels of complexity in terms of number of processors and shared resources.

III. Preliminaries

A. Model of Computation (MoC)

We use synchronous data flow (SDF) as model of computation [12]. This model describes the application data flow between *actors* via communication *channels*. The SDF graph of a Sobel filter example is given in the left upper part of Fig. 1. This MoC offers a strict separation of computation (*Compute*) and communication (*WriteTokens*, *ReadTokens*) statements. Channels are FIFO buffers that can be mapped to any memory that is accessible by the producer and the consumer. The amount of tokens that a producer writes or a consumer reads is called *token rate*. It is denoted by integer values on the SDF graph of Fig. 1. As FIFO accesses are blocking, an actor can only switch to its computation phase after it reads all tokens from all its incoming channels. After the computation phase, the actor switches into write phase to write all tokens to its outgoing channels.

B. Model of Architecture (MoA)

We adopt a tile-based platform organization where each tile consists of a processing element (PE) with local instruction and data memories, and a communication interface (IF). An execution platform consists of a finite set of tiles, a finite set of shared memories and a shared communication bus. In the scope of our work, we consider a first-come-first-served (FCFS) bus arbitration protocol. Shared memory is used to store channel buffers exchanged between the actors of the application. In the example of Fig. 1, the five channels of the modeled application, denoted C_0 to C_4, are thus exchanged through the communication bus and the shared memory. In the scope of this paper, we consider two platform configurations. The first configuration, illustrated in Fig. 1, is a fully composable organization for which each PE uses private instruction and data memories. In that case, a tile can thus execute actor code without interfering with other tiles as long as the software does not explicitly access shared resources. In the second configuration, we considered instruction and data caches for each PE with an additional L2 shared memory connected to all PEs. This organization causes additional variations on software execution time both for computation and communication phases.

C. Measurement Infrastructure

For our performance models we need to characterize the duration of the computation phase of each actor as well as the duration of the communications between actors. For the computation phase, we measure the amount of clock cycles a phase needs to be executed. The measurement infrastructure we use for the computation phase was presented in details in [13]. It can be individually started and stopped from any tile without interference at shared resources. For the communication phases, the adopted measurement infrastructure is able to monitor and store the signals that are relevant to estimate the communications duration. It is used to measure the duration of each elementary state involved in communication and delays caused by contention at shared resources.

D. Statistical model checking methods

Statistical Model Checking (SMC) is used to control simulations of created probabilistic models of studied systems. SMC refers to a series of simulation-based techniques that are used to answer two types of question [14]: (i) What is the probability that a system $\mathcal{S}$ satisfies a property 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 $Pr_{\geq\theta}(\mathcal{S} \models \varphi)$? SMC simply requires an executable model of the system and properties expressed in a classical temporal logic. The core idea of SMC is to monitor a finite set of simulation traces which are randomly generated by executing $\mathcal{S}$. Then, statistical algorithms are 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* [15] to answer the quantitative question and *Sequential Probability Ratio Test (SPRT)* [16] for the qualitative one.

In the case of a quantitative analysis, let $p = Pr(\mathcal{S} \models \varphi)$ be the probability that the system satisfies a certain property φ and p' an estimation of p. Given a precision δ, an estimation procedure consists on computing an approximation p' such that $|p' - p| \leq \delta$ with *confidence* $1 - \alpha$, 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

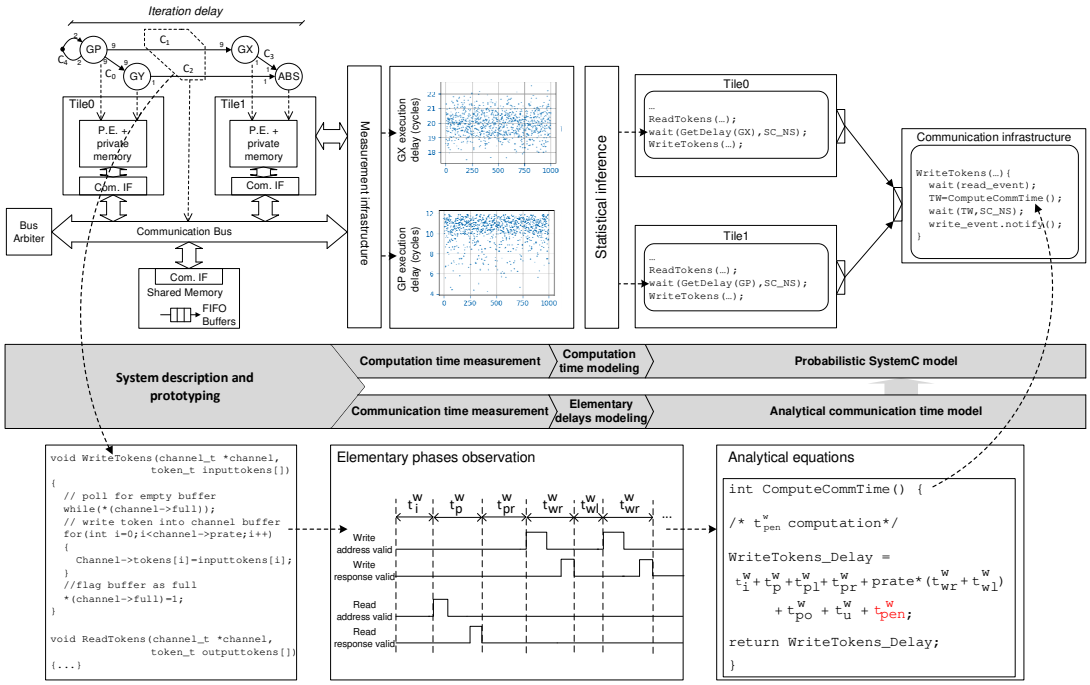


Fig. 1. The left upper part of the figure illustrates the studied systems with applications described as SDFGs and a tile-based platform. The middle upper part illustrates the characterization process of actors computation times. The right upper part illustrates the created SystemC model annotated with timings inferred from measurements. The lower part of the figure represents the communication time modeling process.

is 1 if the simulation satisfies φ and 0 otherwise. 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*. 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 [16], 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). $[p_0, p_1]$ is referred to the *indifference region* of the test which contains θ.

These algorithms are implemented in a set of existing tools such as UPPAAL-SMC [6] and PRISM [17]. In the scope of this work, we especially considered the Plasma-lab statistical model checker [10] as it offers the capability to work directly with SystemC source codes.

IV. Proposed System-level Modeling and Simulation Approach

A. Measurement-based Probabilistic Modeling Approach

1) Computation modeling: The upper part of Fig. 1 illustrates the modeling process of application actors' computation times. Computation times are measured from application execution on platform resources. To allow multiple execution paths of the software to be executed and observed, multiple iterations of the application are run with different input data. A statistical inference process is then led to model the variation of the measured computation delays as a probabilistic distribution function. The parameters of the distribution function are derived from the measured delays. In the case of the first platform configuration, this computation time model is independent from the mapping because of the strict separation between computation and communication statements. In the case of the second platform configuration, this assumption is not valid and the measurement process must be repeated for each studied mapping.

The right upper part of Fig. 1 represents the organization of the created SystemC model. The modules denoted as *Tile0* and *Tile1* describe the execution of the actors mapped on each PE. The timing behavior of a PE is described in an `SC_THREAD` of the *Tile* module. The computation phase of each actor is described with a `wait` statement calling the *GetDelay* function. During each simulated iteration of the system model, this function is used to set the computation time following the distribution function that represents the measured computation delays.

2) Communication modeling: The lower part of Fig. 1 represents the modeling process of communication times between actors. The left lower part gives the typical used code for the *WriteTokens* function. The measurement infrastructure allows elementary phases durations to be measured for multiple iterations of the application. As an example, the elementary

delay denoted as t_p^w corresponds to the duration of polling the buffer state and t_{wr}^w corresponds to the duration of writing one token to the shared memory. These measured elementary delays are used in an analytical model that gives the duration of tokens communication according to the number of ongoing communications. In the equation given in Fig. 1, parameter $prate$ corresponds to the number of tokens and parameter t_{pen}^w corresponds to the introduced delay due to potential contention at communication shared resources. Value of t_{pen}^w depends on the number of ongoing communications.

In the right upper part of the figure, the module denoted as *Communication infrastructure* describes the influence of the shared bus and memory on communications between actors. During simulation, the communication duration is computed at runtime according to the number of ongoing communications taking into account possible contention situations. Further details about the used analytical model and the communication infrastructure description can be found in [18].

B. Simulation-based Analysis Approach with SMC methods

The generation process of monitored SystemC models is illustrated in Fig. 2. A first input of this process corresponds to the created probabilistic system model. A set of variables of interest, called *observed variables*, is defined by the user according to the properties that need to be verified. The observed variables are then used to generate a monitor description with the use of the *Monitor and aspect-advice generator* (MAG) tool introduced in [9]. Afterward, the generated monitor model and the probabilistic system model are instrumented using AspectC++, based on the techniques presented in [19]. The instrumentation is done to allow communication between both models. The two instrumented models are then compiled and linked to the libraries of a patched version of SystemC to build an executable model. This patched version facilitates the communication between the simulation kernel and the monitor and implements a random scheduler for the kernel [20].

In the simulation phase, Plasma-lab iteratively triggers the executable model to run simulations. The generated monitor observes and delivers a set of execution traces to Plasma-lab. An execution trace contains the values of the observed variables and their related timestamp. The timestamping method depends on the resolution parameter used in MAG. In Plasma-lab, properties are expressed with bounded linear temporal logic (BLTL). Properties expressed in BLTL are used by the checker to verify the execution traces. The required number of execution traces is computed by the supported statistical algorithms in Plasma-lab (e.g., Monte Carlo with Chernoff-Hoeffding bounds or SPRT). Users set the parameters of the statistical algorithms to control the confidence level of the analyzed results. The obtained results are the probabilities that the model satisfies the defined properties.

V. Experiments

We aim to evaluate the accuracy and simulation runtime achieved with SMC methods. Two applications were considered with different communication and computation loads. For every considered applications and mappings, we predicted the average iteration delay of the application. We executed all these experiments on a real platform and measured the actual execution times of each iteration. The obtained predictions were compared between simulation and measured results.

A. Experiment definition

Fig. 3 represents the two data flow applications considered to validate our proposed approach. The Sobel filter presents high communication loads whereas the JPEG decoder demonstrates high computation loads. For both applications, we used a noise image of 48x48 pixels as input of these experiments. A token represented a pixel and was coded over 32 bits. Different execution paths could be observed leading to various possible computation times for actors.

Sobel-Filter: For the *GetPixel* actor, there were 9 execution paths depending on the position of pixels in the 3x3 pixels portion of the image processed at each iteration of the application. The *ABS* actor had 4 execution paths depending on the input tokens. There was only one execution path for *GX* and *GY*.

JPEG Decoder: This application presented a huge computation part with an unmanageable amount of execution paths in most of the actors except for the IQ-actors that contained only one execution path.

In Fig. 4, we present the heterogeneous multiprocessor platform that was used for all the experiments. The platform contains 7 tiles with private caches, two shared memories and two shared interconnects. The tiles connect to a DDR shared memory (SM1 in Fig. 4) via BUS1 and exchange data on a BRAM shared memory (SM2 in Fig. 4) via BUS0. Caches can be activated and used to temporarily store data and instructions that are mapped on the DDR shared memory. The measurement infrastructure is presented on the lower part of Fig. 4. Each individual tile uses a MicroBlaze soft core with configurable hardware components. The tiles denoted by *Tile1*, *Tile2*, *Tile3* were extended by a hardware multiplication unit (MUL). Tiles *Tile4*, *Tile5*, *Tile6* were extended with a floating point unit (FPU). The two shared buses applied the protocol AMBA AXI [21].

We considered two platform configurations in the experiments. The first configuration was a composable organization for which each MicroBlaze was equipped with private instruction and data memories. The DDR shared memory and caches were not activated. The BRAM shared memory was used to store exchanged data. Different mappings were applied for each use-case as shown in Tab. I. For the first hardware configuration, we considered 3 mappings for Sobel Filter: *Sobel1a*, *Sobel2a* and *Sobel4a*. For JPEG Decoder, we also did 3 mappings: *Jpeg1a*, *Jpeg3a* and *Jpeg7a*. For example, the experiment denoted by *Sobel2a* corresponds to the mapping of application Sobel-Filter on *Tile1* and *Tile2*.

The second configuration was a non-composable organization for which private caches and DDR shared memory were activated. We considered 3 mappings for Sobel Filter: *Sobel1b*, *Sobel2b* and *Sobel4b*. In these 3 experiments, the instructions and local data of actors were mapped on the DDR

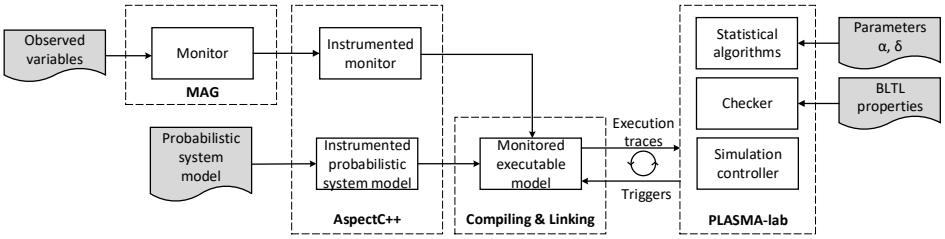


Fig. 2. Generation process of a monitored executable model simulated with SMC methods. At the input of this process, the probabilistic system model has been created following the approach illustrated in Fig. 1.

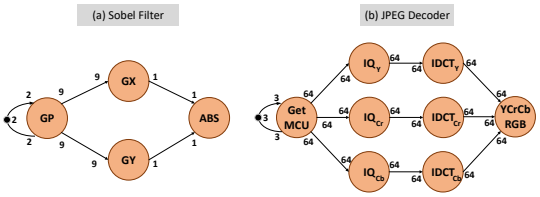


Fig. 3. SDFG applications used in our experiments: (a) Sobel Filter and (b) JPEG Decoder.

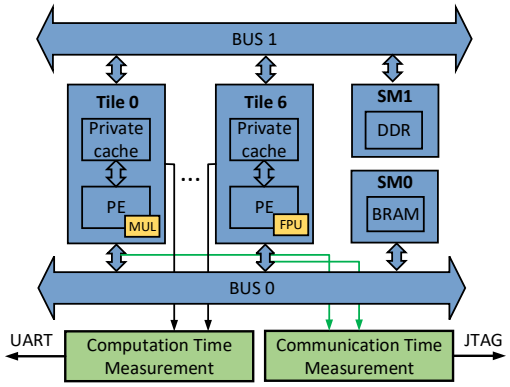


Fig. 4. Heterogeneous platform with 7 tiles with private caches connected to shared memories via shared interconnects. The shared data between tiles are mapped to the BRAM shared memory. In green, the measurement infrastructure contains the computation and communication time measurement.

shared memory. The influences to the execution time caused by cache misses and interferences on accessing the DDR shared memory could be observed for this platform configuration. The JPEG application was not evaluated on this configuration because of excessive execution time.

B. Experiment setup

In our experiments, we used the Xilinx ZC702 Evaluation Board as the prototyping board. The measurement infrastructure consists of two main components: *Computation Time Measurement* and *Communication Time Measurement* which were used to characterize the computation and communication phases of each actor (bottom of Fig. 4). The *Computation Time Measurement* is basically a counter synchronous with the processors (all running at 100 MHz). The counter can be started and stopped from any tile individually without

TABLE I
MAPPING OF THE SOBEL-FILTER AND JPEG EXPERIMENTS ON THE TILES OF THE HARDWARE PLATFORM.

Exp.→ Actor ↓	Jpeg1a	Jpeg3a	Jpeg7a	Exp. → Actor ↓	Sobel1a,b	Sobel2a,b	Sobel4a,b
Get MCU	0	0	0	**GetPixel**	1	1	1
IQ_Y	0	1	1	**GX**	1	2	2
IQ_{Cr}	0	1	2	**GY**	1	1	3
IQ_{Cb}	0	1	3	**ABS**	1	2	0
$IDCT_Y$	0	4	4				
$IDCT_{Cr}$	0	4	5				
$IDCT_{Cb}$	0	4	6				
YCrCb RGB	0	0	0				

interferences with other platform resources. When the counter got stopped, it sends its counter value via the UART to a host computer. Further details can be found in [13].

As the first platform configuration is fully composable, computation time is independent from mappings. We only need to characterize the computation time once. For the second configuration, the computation time of actors needs to be characterized for each mapping. In both cases, the measured computation delays are fitted using the Gaussian distribution.

The elementary delays of the communication phases are characterized using the Xilinx System Integrated Logic Analyzer (SystemILA). For the first configuration, we observed that elementary delays were constant (values are given in Tab. II). For the second hardware configuration, variations were observed in some communication phases. These variations can be explained because of possible cache misses and accesses to the DDR memory when running instructions related to communication code. In Tab. II, we report the observed Best-Case and Worst-Case [BC, WC] of the elementary delays. We used the uniform distribution to represent the variation of these delays in the analytical model adopted to predict communication durations of tokens. Some constant elementary delays were also observed, such as: the polling delay (t_p), the inter-polling delay (t^r_{pl}, t^w_{pl}), the update buffer usage delay (t^r_u, t^w_u), the read delay (t^r_{rd}) and the write delay (t^w_{wr}).

The simulations were done on an Ubuntu PC core i7 2.50 GHz with 8 GB of RAM.

C. Analysis results for the first platform configuration

1) Quantitative analysis: Quantitative analysis is used to estimate the average iteration delay for different mappings. The observed variable $t\_latency$ is added in the created

TABLE II
ELEMENTARY DELAYS (IN CYCLES) MEASURED ON THE 7-TILE PLATFORM. DELAYS WITH UPPER INDEX r ARE RELATED TO READ PHASES, ONES WITH INDEX w ARE RELATED TO WRITE PHASES.

Delay	Config.1	Config.2	Delay	Config.1	Config.2
t_i^r	15	[146, 157]	t_i^w	16	[146, 157]
t_p^r	8	8	t_p^w	8	8
t_{pl}^r	7	7	t_{pl}^w	7	7
t_{pr}^r	15	[13, 66]	t_{pr}^w	15	[43, 86]
t_{rd}^r	8	8	t_{wr}^w	5	5
t_l^r	14	[43, 86]	t_l^w	13	[12, 65]
t_{po}^r	11	[12, 27]	t_{po}^w	9	[9, 36]
t_u^r	5	5	t_u^w	5	5

SystemC model to establish the successive values of application iteration delay. Basically, quantitative analysis is used to estimate the probability that $t\_latency$ stays in a given time interval. This estimation is then repeated for different intervals. The obtained results establish a probability distribution of the iteration delay from which the average value can be estimated. The studied property can be expressed in BLTL as follows:

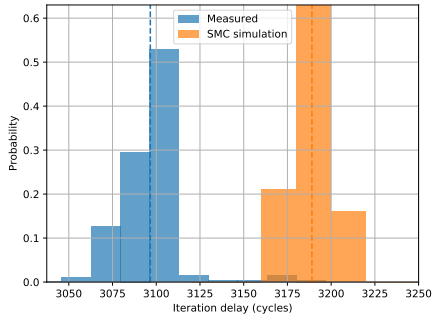
$$declare\ var := [min; max; inc]\ end$$
$$F \leq T(t\_latency \geq var)\&(t\_latency < var + inc)$$

In this BLTL expression, *var* is a variable that stays in a range of values [min, max]. A set of BLTL formulas is generated in Plasma-lab that are then checked during simulation. Plasma-lab estimates thus the probability that the iteration delay $t\_latency$ stays in the time interval [var, var+inc]. Plasma-lab verifies the property in a temporal bound T which is either the number of simulation states or a real-time bound. The modal operator F means that the property is eventually satisfied. We used the Monte Carlo algorithm with Chernoff-Hoeffding bound with absolute error $\delta = 0.02$ and confidence $1 - \alpha = 0.98$. 5757 simulation runs were done for each experiment. Fig. 5 depicts the results of the quantitative analysis for *Sobel4a* and *Jpeg7a* experiments. In both cases, the distributions of simulated results show similar shape compared to the measured data. These distributions are then used to establish the average iteration delay.

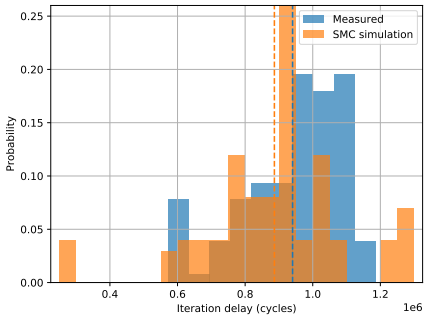
Tab. III compares the average iteration delays obtained with different methods. The second column gives the average iteration delay obtained from the measurement on the prototyping board. In the next column, the average iteration delay of 1 000 000 simulation runs for the basic system model without using any instrumentation or SMC analysis is presented. In the last column, the average iteration delay is obtained using the SMC analysis of the simulation model. The errors between the SMC analysis results and the measured results are shown next to the simulated average iteration delays. The negative error represents an under-estimation compared to the measured data. Otherwise, it shows an over-estimation.

In the Sobel filter experiment, the results of 1 000 000 simulation runs show a good accuracy which is an over-approximation of around 3%. The SMC results present the same level of accuracy as the simulation model without SMC but with limited number of simulation runs. For the JPEG decoder experiments, the SMC results show under-estimations compared to the measured data. The results of 1 000 000 simulation runs show accuracy up to 4.75% in the *Jpeg7a* experiment compared to the measured results and the results of SMC method presents a slight higher error.

In Tab. IV, we compare the simulation duration of the experiments to the measured duration. In the first column, the measured duration of 1 000 000 iterations of the applications running on the real hardware platform is presented. The simulation duration of 1 000 000 iterations without using Plasma-lab is shown in the next column. It took from 4 seconds to around 30 seconds. We then show the simulation time of the SMC analysis. For each SMC experiment, 5757 simulations



(a) Sobel4a, inc = 50



(b) Jpeg7a, inc = 50000

Fig. 5. Distribution of the measured data (blue) compared to the results of the probabilistic simulation (orange) for all experiments. The dashed lines show the median of the distribution.

TABLE III
COMPARISON OF THE SMC ANALYSIS AND THE SIMULATION OF 1 000 000 ITERATION RUNS WITHOUT SMC WITH THE MEASUREMENT DATA IN THE FIRST HARDWARE CONFIGURATION. THE TABLE SHOWS THE AVERAGE EXECUTION TIME (IN CYCLES) AND THE ERROR TO THE MEASURED DATA.

Exp.	Measured	1M analysis	SMC analysis
Sobel1a	3690	3797.5 (2.91 %)	3799.2 (2.96 %)
Sobel2a	2902.5	2977.5 (2.58 %)	2977.8 (2.58 %)
Sobel4a	3097.4	3194.5 (3.13 %)	3189.0 (2.96 %)
Jpeg1a	2385860	2382722.5 (-0.13 %)	2376010.1 (-0.42 %)
Jpeg3a	940836	912219.5 (-3.05 %)	911764.7 (-3.1 %)
Jpeg7a	941059	896367.7 (-4.75 %)	886538.5 (-5.8 %)

TABLE IV
SIMULATION DURATION (IN HH:MM:SS) OF SMC ANALYSIS COMPARED TO THE SIMULATION OF 1 000 000 ITERATION RUNS WITHOUT SMC AND THE MEASURED DURATION IN THE FIRST HARDWARE CONFIGURATION.

Experiment	Measured	1M analysis	SMC analysis
Sobel1a	0:07:23	0:00:04	0:00:29
Sobel2a	0:07:03	0:00:05	0:01:39
Sobel4a	0:07:13	0:00:05	0:01:44
Jpeg1a	13:14:31	0:00:07	0:00:39
Jpeg3a	5:12:58	0:00:12	0:01:52
Jpeg7a	5:13:02	0:00:29	0:02:16

TABLE V
ANALYSIS OF THE *Jpeg7a* EXPERIMENT USING DIFFERENT VALUES FOR THE PARAMETERS OF THE MONTE-CARLO ALGORITHM.

α	δ	Simulation runs	Simulation time(s)	Prob.
0.01	0.01	26492	120.9	0.147
0.01	0.02	23026	102.4	0.075
0.01	0.03	20999	99.4	0.154
0.01	0.04	19561	93.3	0.182
0.01	0.05	18445	93.2	0.213
0.02	0.01	6623	33.2	0.112
0.03	0.01	2944	18.9	0.138
0.04	0.01	1656	9.6	0.126
0.05	0.01	1060	9.6	0.179

were done using Monte-Carlo algorithm ($\delta = \alpha = 0.02$). All the experiments took from around 30 seconds to 2 minutes. The higher simulation time using Plasma-lab can be explained because the execution trace is updated every time a call to the simulation kernel occurs (typically after a `wait` statement). Despite this limitation, the presented approach allows an evaluation that is highly faster than executing applications on a real platform with acceptable level of accuracy.

2) Influences of the parameters: We aim to study the influence of the Monte-Carlo algorithm on the simulation results by changing its parameters. In Tab. V, we present the analysis of the *Jpeg7a* experiment with the Gaussian distribution using different sets of parameters. We defined a property to express the probability that the iteration delay of the *Jpeg7a* experiment stays in the interval [950 000, 1 000 000] cycles. For each set of parameters, we captured the number of simulation runs, the simulation time and the probability to satisfy this property. As we can see in the first half of Tab. V, we first kept the parameter α constant and increased the absolute error δ. In these cases, the number of simulation runs and the simulation time slightly decreased. The obtained probability also slightly changed. Afterward, we kept the absolute error δ constant and decreased the confidence $1 - \alpha$. The number of simulation runs and the simulation time decreased quickly with low influence on the obtained probabilities. The confidence parameter has much more influence on the number of simulation runs compared to the absolute error. This gives a way to control the simulation effort according to the expected confidence in predictions.

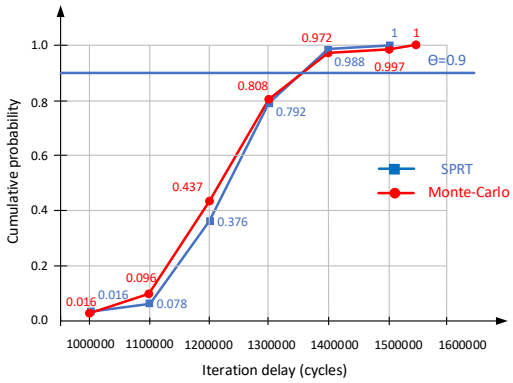


Fig. 6. Cumulative probability of the *Jpeg7a* experiment. The simulation is done using Monte Carlo and SPRT. The probability values are illustrated in red for Monte Carlo and blue for SPRT.

3) Qualitative analysis: We use the qualitative analysis to verifying if the probability to satisfy a given property is greater or equal to a threshold θ. If it is the case, Plasma-lab returns TRUE, and FALSE otherwise. The given property can be expressed in BLTL as follows:

$$declare\ var := [min; max; inc]\ end$$
$$G \leq T(t\_latency \leq var)$$

The variable *var* is incremented by a value *inc* in the range [min, max]. We estimate the probability that the property is always satisfied by using the modal operator *G*. In this experiment, we considered the temporal bound *T* as the duration to finish 100 iterations of the application. It means that the property is satisfied if all the iterations delays are less or equal to the time bound. We used the two statistical algorithms introduced above to estimate probabilities: Monte Carlo with Chernoff bound (with $\delta = \alpha = 0.02$) and SPRT (with $\alpha = \beta = 0.001, \delta = 0.01, \theta = 0.9$). The evolution of the cumulative probability is illustrated in Fig. 6 for the *Jpeg7a* experiment. We found a similar evolution for both cases. When the cumulative probability converges to 1, we found a simulated worst-case iteration delay of 1 550 000 cycles for the *Jpeg7a* experiment with Monte Carlo and 1 500 000 cycles with SPRT. Plasma-lab supports parallel simulation with multi-threaded execution. In these experiments, we used 4 threads to reduce the simulation time. The analysis time to capture all these probabilities is 10 minutes for Monte Carlo while SPRT showed a faster simulation time of 4 minutes.

D. Analysis results for the second hardware configuration

In this section, we consider the application of SMC methods to the Sobel filter application on the second platform configuration. In this experiment, the use of cache caused higher variability of the communication time. For the SMC analysis, we also analyzed the *probability distribution* using the Monte Carlo algorithm with Chernoff-Hoeffding bound (with the absolute error $\delta = 0.02$ and the confidence $1 - \alpha = 0.98$). In Tab. VI, we compare the average iteration delay obtained from

TABLE VI
COMPARISON OF THE SMC ANALYSIS AND THE SIMULATION OF 1 000 000 ITERATION RUNS WITHOUT SMC WITH THE MEASURED DATA IN THE SECOND HARDWARE CONFIGURATION. THE TABLE SHOWS THE AVERAGE EXECUTION TIME (IN CYCLES) AND THE ERROR TO THE MEASURED DATA.

Experiment	Measured	1M analysis	SMC analysis
Sobel1b	20634.2	20547.5 (-0.42 %)	20529.2 (-0.51 %)
Sobel2b	15533.8	15459.7 (-0.48 %)	15479.0 (-0.35 %)
Sobel4b	15455.1	14907.7 (-3.55 %)	15321.9 (-0.87 %)

measurement with the simulation results of 1 000 000 iterations and the SMC results. The iteration delays in these experiments are much higher than in the first hardware setup due to the longer access time to instructions and data mapped on the DDR memory. The simulation results of 1 000 000 iterations show a good accuracy in all experiments and a maximum error of 3.55% in the *Sobel4b* experiment. We found similar results on accuracy using the SMC approach. However, the simulation time of the experiments without SMC took around 5 seconds, while the SMC experiments took more than one minute.

E. Discussion

Performance prediction with SMC methods showed a good level of accuracy but increased the simulation duration compared to a model simulated without SMC for the two hardware configurations. The increase in simulation duration can be explained by the fact that the monitoring method leads to many execution traces to be generated which causes much longer simulation time compared to a pure SystemC execution. This could represent a limitation of the approach in case of more complex system models. However, the presented approach demonstrates capabilities for accurate and controlable quantitative and qualitative analysis of created models both for composable platforms and platforms with hierarchy of shared resources.

VI. CONCLUSION

In this paper, we evaluated the benefits of SMC methods for performance estimation of multiprocessor systems. SMC methods allow the number of simulation runs to be controlled with still good level of accuracy in predictions. In the case of composable platforms, this makes possible to explore and compare various mappings with limited evaluation effort. However, further works are still needed to reduce the characterization effort in the case of non-composable platforms and to favor the creation of scalable models.

ACKNOWLEDGMENT

This work has been partially sponsored by We Network and Campus France (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).

REFERENCES

[1] J. Rutten, M. Kwiatkowska, G. Norman, and D. Parker, *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems,* P. Panangaden and F. van Breugel (eds.), ser. CRM Monograph Series. American Mathematical Society, 2004, vol. 23.

[2] K. G. Larsen and A. Legay, "Statistical model checking: Past, present, and future," in *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques*, T. Margaria and B. Steffen, Eds. Cham: Springer International Publishing, 2016, pp. 3–15.

[3] 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.

[4] A. Nouri, M. Bozga, A. Molnos, A. Legay, and S. Bensalem, "Astrolabe: A rigorous approach for system-level performance modeling and analysis," *ACM Trans. Embed. Comput. Syst.*, vol. 15, no. 2, Mar. 2016. [Online]. Available: https://doi.org/10.1145/2885498

[5] 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.

[6] A. David, K. G. Larsen, A. Legay, M. Mikučionis, and D. B. Poulsen, "UPPAAL SMC tutorial," *International Journal on Software Tools for Technology Transfer*, vol. 17, no. 4, pp. 397–415, 2015.

[7] 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.

[8] R. Stemmer, H. Schlender, M. Fakih, K. Grüttner, 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)*, 2019, pp. 1715–1720.

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

[10] 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.

[11] R. Stemmer, H.-D. Vu, K. Grüttner, S. Le Nours, W. Nebel, and S. Pillement, "Experimental evaluation of probabilistic execution-time modeling and analysis methods for SDF applications on MPSoCs," in *2019 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. Springer, 7 2019, pp. 241–254.

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

[13] 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.

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

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

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

[17] 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.

[18] H.-D. Vu, S. L. Nours, S. Pillement, R. Stemmer, and K. Grüttner, "A fast yet accurate message-level communication bus model for timing prediction of sdfgs on mpsoc," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '21, 2021, p. 17–22.

[19] D. Tabakov and M. Y. Vardi, "Monitoring temporal systemc properties," in *Eighth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE 2010)*. IEEE, 2010, pp. 123–132.

[20] ——, "Automatic aspectization of systemc," in *Proceedings of the 2012 workshop on Modularity in Systems Software*, 2012, pp. 9–14.

[21] A. Limited, "Amba® axi™ and ace™ protocol specification," aXI3, AXI4, and AXI4-Lite ACE and ACE-Lite.