Experimental Evaluation of Probabilistic Execution-Time Modeling and Analysis Methods for SDF Applications on MPSoCs

Ralf Stemmer, Hai-Dang Vu, Kim Grüttner, Sébastien Le Nours, Wolfgang Nebel, Sébastien Pillement

Experimental Evaluation of Probabilistic Execution-Time Modeling and Analysis Methods for SDF Applications on MPSoCs *

Ralf Stemmer[1], Hai-Dang Vu[2], Kim Grüttner[3], Sebastien Le Nours[2], Wolfgang Nebel[1], and Sebastien Pillement[2]

[1] University of Oldenburg, Germany {ralf.stemmer, wolfgang.nebel}@uol.de
[2] University of Nantes, IETR, UMR CNRS 6164, France {hai-dang.vu, sebastien.le-nours, sebastien.pillement}@univ-nantes.fr
[3] OFFIS e.V., Germany kim.gruettner@offis.de

Abstract. Early validation of software running on multi-processor platforms is fundamental to guarantee that real-time constraints will be fully met. In the domain of timing analysis probabilistic simulation techniques tackle the problem of scalability. However, creation of probabilistic SystemC models remains a difficult task and is not well supported for multi-processors systems. In this paper we present a modeling workflow that will then be used for an experimental evaluation of probabilistic simulation techniques. For the modeling process a measurement-based approach is proposed to favor the creation of trustful models. The evaluated probabilistic simulation techniques demonstrate good potential to deliver fast yet accurate estimations for multi-processor systems.

Keywords: Statistical Model Checking · Probabilistic SystemC Model · Multi Processor

1 Introduction

Multi-processor systems are increasingly adopted to implement high performance time critical systems. In the design of such systems, early verification that real-time constraints are fully met is fundamental to prevent costly design cycles. Timing analysis of parallel software running on multi-processors is hard, especially because of possible interferences among application tasks due to contention at the shared resources of the processor (*i.e.*, communication bus, shared memory, shared caches). In this context, appropriately capturing and analyzing low-level influences of platform shared resources early in the design process represents

* 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)

a challenging effort. Existing real-time analysis methods, *i.e.*, simulation-based and formal mathematical approaches, show limitations to deliver fast yet accurate estimation of timing properties.

Probabilistic models represent a possible solution to capture variability caused by shared resources on parallel software execution [14]. 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 [11]. In the field of embedded system design, executable specifications built with the use of the SystemC language are now widely adopted [1]. 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-processor systems is promising because it could deliver a good compromise between accuracy and analysis time, yet it requires a more sophisticated timing model based 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-processor systems with complex hierarchy of shared resources, exploring their application on trustful probabilistic SystemC models remains a significant research topic.

In this paper, we present an experimental modeling setup that is used to evaluate the efficiency of SMC methods for multi-processor systems. The contributions of this paper are 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-processor 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 setup on a Sobel filter case study. Two configurations of the hardware platform with different levels of complexity are considered to analyze the relevance and effectiveness of SMC methods.

This paper is organized as follows. In Section 2 we provide an overview and discussion of relevant related work. Section 3 presents the established modeling and analysis approach. The experimental results are described in Section 4. Section 5 discuss the benefits and limitations of the presented approach.

2 Related Work

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 require extensive architecture analysis under various possible working scenarios. However, due to insufficient corner case coverage, simulation-based approaches are limited to determine guaranteed limits about system properties.

Statistical Model Checking (SMC) has been proposed as an alternative to formal 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. Various probabilistic model-checkers support statistical model-checking, as for example UPPAAL-SMC [5], Prism [9], and Plasma-Lab [7].

We have presented a comparison of formal model checking with SMC in [17]. A multi-processor system was modeled using timed automata and probabilistic timed automata in UPPAAL-SMC. While formal methods use only best and worst case execution times the SMC method allows to model the distribution of execution times between those limits. The possibilities of modeling such a distribution is limited due to UPPAALs modeling language. In contrast to that work, our new SystemC approach allows to not only use a rough model of the distribution but the actual measured execution times. Furthermore we refined the communication model to better cover the computation overhead on the CPUs that accesses the communication infrastructure.

Authors in [3,4] 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 [8] 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 [15] 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 [2]. 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 the system model is presented in [14] where a statistical inference process is proposed to capture low-level plat-

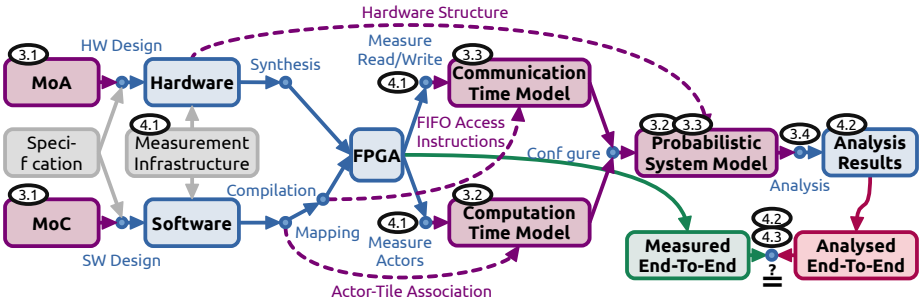


Fig. 1. Overview showing where and how the models introduced in Section 3 are used. The circled numbers reference the sections where more details can be found.

form 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 [13] 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 [13] to analyze time properties of multi-core systems modeled with SystemC.

To the best of our knowledge, no other work attempted to systematically evaluate the benefits of using SMC to analyze the timing properties of SDF based applications on multi-processor systems, targeting more tightness of estimated bounds and faster analysis times.

3 Characterization and Modeling Approach

Fig. 1 shows a detailed insight in our approach, highlighting the models we use (purple rectangle). The circled number above those rectangles references the section where those models are explained in detail. Other references point to the related experiment Sec. 4. We design our system following the hardware (HW) and software (SW) models in Sec. 3.1. The HW and SW designed following those models get extended by a measurement infrastructure introduced in Sec. 4.1 and described in detail in [16]. The mapped and scheduled software gets mapped and executed on the designed hardware that gets instantiated on an FPGA. This *real system* is used to characterize the timing behavior of the HW components as well as the software. The insight in the instantiated system allows modeling the communication in detail, considering the computational overhead by the software (derived from its instructions) that manages the communication between shared resources (See Sec. 3.3). For the computational part of our application, we can model its timing behavior on a specific hardware in detail as described

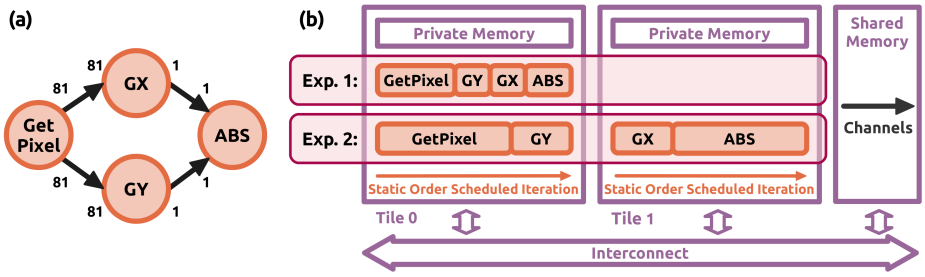


Fig. 2. a) A Sobel filter modeled as an SDFG. A pixel matrix gets processed in *GX* and *GY* and returned to the *ABS* actor that calculates the resulting pixel.
b) Our platform consisting of two tiles. In the first experiment, all actors are mapped to *Tile 0*, in the second experiment both tiles are used. The order of the mapped actors represents their scheduling.

in 3.2. The modeled communication and computation time then get integrated into a probabilistic SystemC model. This model then gets used for execution time analysis focusing on the distribution of the execution time. Using the architecture and computation model for designing our system allows us to do some assumptions on our performance models (visualized as dotted lines in Fig. 1). We analyze this model using a statistical model checker as explained in Sec. 3.4. In our experiments (Sec. 4) we compare the analysis results with the observed behavior of the system.

3.1 System Model

This section explains our software and hardware model, as well as the mapping and scheduling of the SW on the HW. These models are the input models we use to constraint our HW platform and our SW. This allows us to do assumptions on the resulting system we can later use to model its timing behavior.

Model of Computation (SDF): The Sobel filter (see Fig. 2a), used in our experiments, follows SDF semantics that was proposed by [10]. 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. The actors are statically scheduled and will not be preempted.

The channels used for communication between actors are implemented as FIFO buffers on a shared memory. For the single processor setup (Fig. 2b, Exp. 1), the communication time will be deterministic because the shared memory is actually only used by a single processor. Beside the data dependencies, and so the communication channels, there is no further synchronization.

On the multi processor setup (Fig. 2b, Exp. 2) application iterations can overlap over time. During the read an write phases the actors are polling on the FIFO states until the buffer is full or empty. Mapping the application onto more than one processor, the communicational parts interfere each other.

Model of Architecture: Our hardware architecture allows us to design a composable multi processor system that uses multiple independent tiles to execute applications as deterministic as possible. A tile consists of one processing element and a private memory only associated to that processing element via a separate bus. It can execute software without interfering with other tiles as long as the software only accesses the private memory. For our example we use a MicroBlaze as processing element.

To improve the determinism for our experiments, we start with a single processor. This allows the assumption that accessing the shared memory (see Fig. 2b) will always be exclusive. Later we add a second processor and change the mapping of the application to add communication interference to the system.

The whole execution platform consists of multiple tiles, buses and shared memories. 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.

All channels buffers are mapped onto shared memory, while the actors are stored on private memory of the tiles they get executed on (Fig. 2b). This setup is fully 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).

3.2 Computation Model

All computation times are related to the mapped software (i.e. SDF actors and channels) on a specific hardware architecture. For this reason, the annotation of any delays (in cycles) to the models takes place after the mapping process as shown in Fig. 3 top left corner.

The execution of an actor's computation phase for a specific architecture can be represented by a single delay $d \in \boldsymbol{D}_c$ for each execution. To model an actor, its execution time gets measured and the delay vector $\boldsymbol{D}_c$ gets derived from the measured values in a way that the distribution of possible delay in $\boldsymbol{D}_c$ follows the distribution measured delays of the characterizing actor. Each element in $\boldsymbol{D}_c$ of an actor is a sample from the measurement. In the SystemC implementation, those values get selected by a `GetDelay` function as shown in Fig. 3 top left part.

3.3 Communication Model

The communication timing model requires more effort compared to the computation model. To cover interference on the shared resources inside the analysis, there are four things to consider:

- The communication between actors includes computational parts for accessing the private memory data and calculating addresses.
- The amount of traffic generated on the interconnect depends on the number of tokens that get communicated.

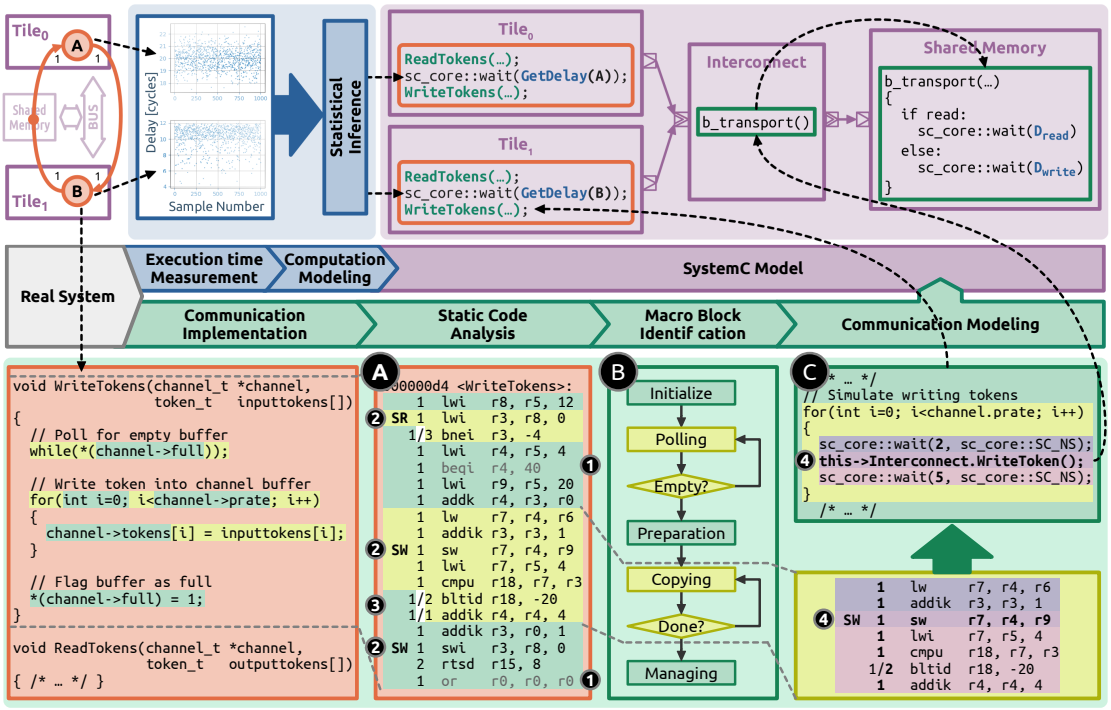


Fig. 3. This figure shows the whole process of creating a SystemC model (purple) from a real system (top left corner). In blue, the approach to model the computation time is showed, in green the process for modeling communication.

– The communication function needs to wait and check repetitively the state of the channels buffer to have tokens or space available (aka polling).
– To model bus contention as precisely as possible, the moment of shared memory access must be as accurate as possible.

Since the communication model is crucial, the behavior of this part is modeled in detail to come as close as possible to the real system. The process to get this model can be applied for many different architectures, we demonstrate it for a MicroBlaze architecture. The whole process is visualized in Fig. 3 on the bottom half using `WriteTokens` as example. The following paragraphs explain the single steps of this process. The same process has been done for `ReadTokens` as well.

Communication Implementation: Communication is done by two functions used in the implemented software. One function is `WriteTokens` that write tokens onto the shared memory (Used as example in Fig. 3 bottom left) and the other function is `ReadTokens` that reads from shared memory. So only those two functions need to be modeled in detail, while all other parts of the SDF application gets represented by its computation time $d \in \boldsymbol{D}_c$ (Sec. 3.2). The functions were analyzed in detail via static code analysis on instruction set level

Static Code Analysis: After disassembling the two communication functions, the amount of clock cycles to execute each instruction needs to be annotated to the code. This requires some understanding of the code and the architecture. Fig. 3 Ⓐ shows the annotated assembly code. There are two instructions marked by a ①. The first instruction is a branch that will never be taken as long as it is guaranteed that the token rate will not be zero. So this branch instruction can be considered a *No Operation* instruction and will only take one cycle. The same is for the last line, also marked as ①. This instruction gets executed due to the pipeline implementation of the MicroBlaze.

Other lines are marked with ②. These lines are commented with `SR` and `SW` to mark instructions that read and write to shared memory. Shared memory access takes more cycles than just the execution of the instruction. These are the points where the simulation of the model needs to consider resource contention.

Macro Block Identification: After annotation the cycles to the instructions, the code gets separated into its macro blocks. In Fig. 3 Ⓑ the execution of the macro blocks is visualized as a flowchart. The colors highlight those blocks in the original source code and the resulting instructions.

Both functions consists of a polling part that depends on the amount of polling iterations n (Fig. 3 Ⓑ upper loop), and a part where the tokens get copied which depends on the amount t of tokens (Fig. 3 Ⓑ lower loop). The amount n of polling iterations is determined during the analysis of the model and varies between each SDF execution iteration i. For a single processor setup with a valid scheduling, the amount of polling will always be one ($n = 1 \; \forall \; i$). The token transfer rate t equals the consume and produce rate of the actors that access the channel.

There may be some instructions that need to be considered for two macro blocks. For example in the instruction listing Fig. 3 Ⓐ mark ③ shows a situation where not only a branch instruction but also the one after is part of the *Copying* block and the *Managing* block. While for the branching instruction it is obvious since it can be taken or not, the next instruction gets executed in both situations due to the architecture specific pipeline implementation.

Communication Modeling: Next, the SystemC model can be built out of the macro blocks. This is shown in Fig. 3 Ⓒ for the *Copying* block. This block must be split into three parts. The 1st and the 3rd parts are the instructions before and after the shared memory access (③) and can be represented by a SystemC `wait` statement. The 2nd part is the shared memory access. The instruction for the shared memory access gets represented by a function call that will trigger the interconnect module of the SystemC system model. The read and write access to a shared resource gets represented by the delay vectors $\boldsymbol{D_r}$ and $\boldsymbol{D_w}$ that is represented the same way as for computation in Sec. 3.2.

By representing all macro blocks with `wait` statements and interconnect accesses, the communication can be modeled in detail. The interconnect module itself can now represent the behavior of the interconnect and the shared memory that is connected to the interconnect.

3.4 Statistical Model-Checking of Probabilistic SystemC Models

Statistical Model Checking (SMC) refers to a series of simulation-based techniques that can be used to answer two types of question [12]: (i) What is the probability that the system satisfies a property (*quantitative analysis*) and (ii) Is the probability that the system satisfies a property greater or equal to a threshold value (*qualitative analysis*)? The core idea of SMC is to monitor a finite set of simulation traces which are randomly generated by executing the probabilistic system. Then, statistical algorithms can be used to estimate the probability that the system satisfies the property. In the scope of this paper, we consider two algorithms: *Monte Carlo* with *Chernoff-Hoeffding bound* (Monte Carlo) for quantitative analysis and *Sequential Probability Ratio Test (SPRT)* for qualitative analysis (see details in [14]). Although SMC only provides an estimation, the algorithms presented below offer strict guarantees on the precision and the confidence of the test.

The statistical model-checker workflow that we consider takes as inputs a probabilistic model written in SystemC, a set of observed variables, a Bounded Linear Temporal Logic property (BLTL), and a series of confidence parameters needed by the statistical algorithms. First, 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 *Monitor and aspect-advice generator* (MAG) tool proposed by V.C Ngo in [13] to generate an aspect-advice file and a monitor model. The aspect-advice file declares the monitor as a *friend* class, so that the monitor can access to the private variables of the observed model. Then, the generated monitor and the probabilistic SystemC model 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.

In the Sobel filter case study, we create the probabilistic SystemC model by using the distributions provided by GNU Scientific Library (GSL) [6]. These distributions represent the variation of the computation time of four actors of the Sobel filter. In the scope of this paper, we use the uniform and normal distributions. In each iteration of simulation of the probabilistic SystemC model, the computation time is assigned to a value that is randomly chosen following the distributions. In the SystemC model, the communication time is represented by `wait` statements. Wait durations depend on the estimated read/write delays and the number of polling states.

To apply the uniform distribution, the computation time is randomly chosen from a value in the interval of [BCET, WCET] of the measured data. While in the normal distribution, we consider the mean μ and the variance σ of the corresponding measured delays. It is desired to create a probability distribution

that accurately reflects the distribution of the measured delays to verify the accuracy of the uniform/normal distributions. This is realized by reading a delay value of every actor from a text file that provides the raw measured computation time of this actor. We refer these raw data to the injected data. The distribution of the randomly selected delay values from the raw measured data file is uniform.

4 Experiments

In this section we describe our experiments and discuss the results. We use a SystemC implementation of our models described in Section 3. The timing behavior got characterized as described in Subsection 3.2 and Subsection 3.3.

4.1 Experiment Definition

In our experiment we successively consider a single processor system and a multi processor system as shown in Fig. 2b. For the single processor setup, all actors of the SDF application shown in Fig. 2a are mapped to *Tile 0*. For the multi processor setup, two actors are mapped to *Tile 0* and two to *Tile 1*. This allows us to do an analysis with and without having to consider bus contention. We then compare the analysis results with focus on the communication model.

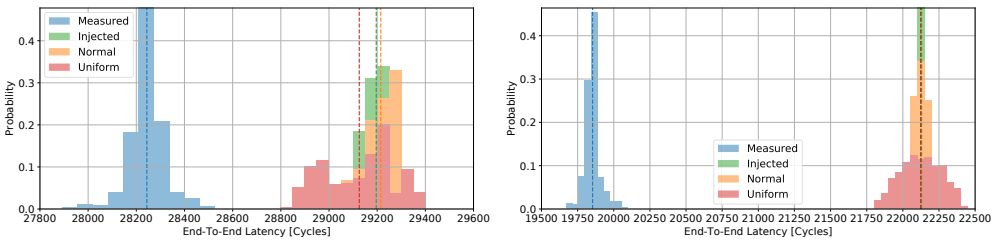
A shared memory is used for the data exchange between actors for both setups. In a multi processor setup, all processors have access to the shared memory, but not to the private memory of the different processors. The interconnect to the shared memory uses First-Come First-Serve arbitration and a Single-Beat transfer protocol. The channels are organized as FIFO buffers with 4 Bytes for each token and have a fixed buffer size.

The measurement technique we use in the experiments to get the different delay vectors of different components is based on one presented in [16]. 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 [16]).

4.2 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 estimated latency as an observed variable *t_latency* in the configuration file used for the SystemC model generation for Plasma Lab. To quantitatively evaluate the 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 F for "eventually": $\varphi = F_{\leq T}((t\_latency \geq d_1)\&(t\_latency < d_2))$.

[tb]



(a) Single processor **(b)** Dual processors

Fig. 4. Analysis results using the different distributions (normal/uniform) or the injected data for the computation time comparing with the measured results in the (a) single processor and (b) dual processors experiments.

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 latency stays in the interval $[d_1, d_2]$. Fig. 4 presents the probability distribution of the analysis results comparing to the measured results of the single processor and dual processors experiments. We use the Monte Carlo algorithm to analyze this property with the absolute error $\delta = 0.02$ and the confidence 98 %.

In the single processor experiment, the normal distribution and the injected data show a similar shape of the distribution to the measured data (Fig. 4a). The uniform distribution shows the same range of variation comparing to the measured data (600 cycles), while a smaller variation (400 cycles) is observed in the case of the normal distribution and the injected data. The uniform data presents the most over-estimation comparing to injected and the normal distributed.

In the dual processors experiment, the analyzed results of the normal distribution and the injected data (Fig. 4b) show the similar range of variation (around 400 cycles) and the shape of the distribution comparing to the measured data. All the analyzed results clearly over-estimate the measured data and the uniform distribution still shows a more pessimistic over-estimation.

The fact that the uniform distribution can show a more pessimistic over-estimation is because in that case the WC computation time of each actor has a higher probability to be taken into account during the analysis process than in the normal distribution and the injected data.

4.3 Evaluation of Statistical Model Checking Methods

We want now to bound the WC 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 G for "always": $\varphi = G_{\leq T}(t\_latency \leq d)$.

Table 1. Estimation of the WC latency with the analysis accuracy and duration.

Subject		**1 Tile**	**2 Tiles**
Measured data	End-To-End in Cycles	28527	20097
	Uniform	3.55 %	11.71 %
Over-approximation	Normal	3.11 %	11.01 %
	Injected data	3.53 %	11.51 %
	Monte Carlo (5757 simulations)	4.25 min	12 min
Analysis time	SPRT (342 simulations)	11 sec	22 sec

The temporal modal operator G is applied that allows us to check whether all the latencies of several successive iterations during the simulation time T stay below d. To analyze this property, we apply two statistical algorithms: Monte Carlo (the absolute error $\delta = 0.02$ and the confidence 98 %) and SPRT ($\alpha = \beta = 0.001, \delta = 0.01$). Tab. 1 summarizes the results of the over-approximation of the three cases and the analysis time for the uniform distribution.

In the single processor experiment, the WC latency of the probabilistic SystemC model applying the uniform, normal and injected data is bounded to 29541, 29415 and 29535 cycles, respectively, comparing to 28527 cycles of the WC measured latency. The uniform distribution shows the most pessimistic over-estimated WC latency of 3.55 % compared to the measured data. In the dual processors experiment, we also get the over-estimation of the WC latency in all three cases and the over-estimation results is around 11 %. The uniform distribution still presents the most pessimistic over-estimated results of 11.71 % compared to the measured data.

For each experiment, the statistical algorithms observe the same number of simulation runs (see Tab. 1). Since SPRT observes a smaller number of simulation than Monte Carlo, it takes less analysis time to analyze one property. In the second experiment, the higher analysis time for each iteration leads to a higher overall analysis time compared to the first experiment. In the case that we only want to bound a probability with a threshold value, SPRT is more efficient than Monte Carlo in terms of analysis time.

4.4 Discussion

In the case of experiment with one processor the difference between the measured and approximated latency is acceptable. In the dual processors experiment 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.

The different range of variation in the experiments can be explained by the variation of GX and GY. They have a higher impact on a single processor system because there they sum up. On the dual processors system, their variation are less dominant because of the synchronizing behavior ABS that reduce the impact

of the variation. So the variation of GY execution time does not matter as long as GX has not finished and vice versa.

The bias between the measured and the simulated latency comes from a pessimistic shared memory 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 their symmetry. The ABS actor waits for the slowest dependent actor (GX, GY). In our simulation the delay of GX and GY gets selected independently. Therefore, the measured timings for faster executions get covered by the slower delays.

In the experiments, worst case end-to-end latency was estimated with an approximation close to 3 % and 11 % compared to real implementations. With our old models used inside UPPAAL SMC we got 15 % for 2 tiles [17]. One benefit of our approach lies in the possibility to control the number of simulation runs given a level of confidence. In [17] we showed that a simulation (inside UPPAAL SMC) for low confidence (99.5 %) can be several times faster than analyzing the same model using a formal approach. The results showed that the SPRT analysis of the model for the 2-tile configuration took 22 s. The more abstract Timed Automata model from [17], with a less detailed communication model and with the same analysis configuration, took about 10.3 s on the same CPU.

5 Summary & Future Work

In this work, we have presented a modeling setup that is used to evaluate the efficiency of SMC methods to analyze real-time properties of SDF applications running on multi-processor systems. 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 tile 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. The SPRT simulation method proved significantly reduced analysis time compared to Monte-Carlo. By controlling the number of simulation runs, a trade-off between high confidence and fast analysis time is possible. In future work we will increase the number of tiles to evaluate the scalability of our approach. Additionally we want to improve our model by considering data dependencies since SystemC allows us to also do a functional simulation of our system.

References

1. Association, I.S., et al.: Ieee standard for standard systemc language reference manual. IEEE Computer Society (2012)
2. Basu, A., Bensalem, B., Bozga, M., Combaz, J., Jaber, M., Nguyen, T., Sifakis, J.: Rigorous component-based system design using the bip framework. IEEE Software **28**(3), 41–48 (May 2011). https://doi.org/10.1109/MS.2011.27

3. Cazorla, F.J., Abella, J., Andersson, J., Vardanega, T., Vatrinet, F., Bate, I., Broster, I., Azkarate-Askasua, M., Wartel, F., Cucu, L., et al.: Proxima: Improving measurement-based timing analysis through randomisation and probabilistic analysis. In: Digital System Design (DSD), 2016 Euromicro Conference on. pp. 276–285. IEEE (2016)
4. Cazorla, F.J., Quiñones, E., Vardanega, T., Cucu, L., Triquet, B., Bernat, G., Berger, E., Abella, J., Wartel, F., Houston, M., et al.: Proartis: Probabilistically analyzable real-time systems. ACM Transactions on Embedded Computing Systems (TECS) **12**(2s), 94 (2013)
5. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B., van Vliet, J., Wang, Z.: Statistical model checking for networks of priced timed automata. In: Fahrenberg, U., Tripakis, S. (eds.) Formal Modeling and Analysis of Timed Systems, Lecture Notes in Computer Science, vol. 6919, pp. 80–96. Springer Berlin Heidelberg (2011)
6. GSL: https://www.gnu.org/software/gsl/
7. Jegourel, C., Legay, A., Sedwards, S.: 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). pp. pp.498 – 503 (2012)
8. Kumar, A.: Analysis, design and management of multimedia multi-processor systems. Ph.D. thesis, Eindhoven University of Technology (2009)
9. Kwiatkowska, M., Norman, G., Parker, D.: Prism 4.0: Verification of probabilistic real-time systems. In: In Proc. International Conference on Computer Aided Verification (CAV'11). pp. 585–591 (7 2011)
10. Lee, E.A., Messerschmitt, D.G.: Synchronous data flow. Proceedings of the IEEE **75**(9), 1235–1245 (1987)
11. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: Barringer, H., Falcone, Y., Finkbeiner, B., Havelund, K., Lee, I., Pace, G., Roşu, G., Sokolsky, O., Tillmann, N. (eds.) Runtime Verification. pp. 122–135. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
12. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. International conference on runtime verification pp. 122–135 (2010)
13. Ngo, V.C., Legay, A., Quilbeuf, J.: Statistical model checking for systemc models. 2016 IEEE 17th International Symposium on High Assurance Systems Engineering pp. 197–204 (2016)
14. Nouri, A., Bozga, M., Moinos, A., Legay, A., Bensalem, S.: 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)
15. Nouri, A., Bensalem, S., Bozga, M., Delahaye, B., Jegourel, C., Legay, A.: Statistical model checking qos properties of systems with sbip. International Journal on Software Tools for Technology Transfer **17**(2), 171–185 (2014)
16. Schlaak, C., Fakih, M., Stemmer, R.: Power and execution time measurement methodology for sdf applications on fpga-based mpsocs. arXiv preprint arXiv:1701.03709 (2017)
17. Stemmer, R., Schlender, H., Fakih, M., Grüttner, K., Nebel, W.: 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)