Probabilistic State-Based RT-Analysis of SDFGs on MPSoCs with Shared Memory Communication

Ralf Stemmer, *University of Oldenburg*, ralf.stemmer@uol.de
Henning Schlender, *University of Oldenburg*, henning.schlender@uol.de
Maher Fakih, *OFFIS e.V.*, maher.fakih@offis.de
Kim Grüttner, *OFFIS e.V.*, kim.gruettner@offis.de
Wolfgang Nebel, *University of Oldenburg*, wolfgang.nebel@uol.de

***Abstract*—This paper extends a state-based timing analysis for Synchronous Dataflow Applications on an MPSoC with shared memory. The existing approach transforms a mapped and timing annotated SDF graph into a timed automata representation for the analysis of timing properties. One major drawback of the existing timing annotation approach is the usage of best- and worst-case execution time intervals, resulting in an overestimation of the actual timing behavior. This paper proposes to replace the timing bound annotation with a probability density function. For the overall timing analysis we use a stochastic timed automata model.**

We demonstrate and evaluate our approach on a Sobel filter, which is used in many image and video processing algorithms. As a reference, we compare our stochastic execution time model against a fixed best-/worst-case execution time model and against the measured execution time on an FPGA prototype.

The results are promising and clearly indicate that our probabilistic approach provides tighter timing analysis results in comparison to the best-/worst-case execution analysis model.

I. Introduction

The increasing complexity of modern embedded systems and the wide distribution of multiprocessor systems-on-chip (MPSoC) are challenging existing traditional real-time analysis methods. Yet, this is an indispensable challenge for guaranteeing their safe usage in safety critical domains (avionics, automotive). Towards coping with such challenges, this paper extends a state-based timing analysis [1] for *Synchronous Dataflow* (SDF) [2] Applications on an MPSoC with shared memory. In that work a mapped and timing annotated SDF graph is transformed into a *Timed Automata* (TA) representation for the analysis of timing properties.

Synchronous Data Flow Graphs (SDFGs) have great analysability and are a common model to capture the behavior of data-driven applications [3]–[5].

We use SDFGs to validate the fulfillment of timing requirements to guarantee the correct time response of the system. The timing analysis uses best- and worst-case execution time (BCET or WCET) intervals. This leads to an overestimation of the actual timing behavior. Furthermore, timing intervals can have a high impact on the timed automata state space and thus analysis time as shown in [1]. This paper proposes to replace the timing bound annotations with a probability density function (PDF). For the overall timing analysis of the mapped SDF application, instead of the classic TA model, a *Stochastic Timed Automata* (STA) model is used in this work.

Timed Automata is a common model for applications to verify their real-time requirements. The STA extension [6] enables the model to have probability functions for expressing the timing behavior. In our work we use STAs capture measured execution times of SDF actors in our timing analysis framework. We use the *UPPAAL* model-checker [7] to verify the real-time requirements of our SDF system.

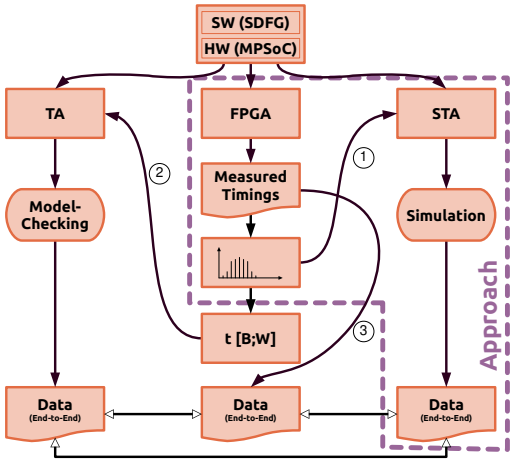


Fig. 1. Overview of our approach to use measured execution time (middle column) in our (dashed) STA analysis (right column) including a comparison with the previous TA approach [1] (left column)

We have evaluated our approach on a Sobel filter case study and compared the obtained end-to-end timing analysis results between the TA model, the STA model and measurements on an FPGA prototype.

Our approach is shown in Fig. 1. 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 Sec III in detail.

We compare our methodology with the approach from [1], which is shown on the left of Fig. 1. In this approach the system is modeled as a TA and analyzed using a static model checker. It uses Best- (BCET) and Worst-Case Execution Time (WCET) intervals ②, which represent safe lower and upper bounds of the measured (i.e. observed) execution timed on the specific hardware platform.

In our new STA approach we use a more sophisticated timing model, based on probability density functions, which are inferred from the measured execution times of each actor on the used hardware platform (see arrow ① in Fig. 1). This allows a more realistic analysis and does not hide important information (e.g. the average execution time, execution time peaks or outlier). Furthermore, our new approach can also generate a distribution of the resulting end-to-end delay. This can provide very relevant information soft-real time systems or further analysis based on probabilistic execution time analysis.

While the TA-based model-checking approach [1] quickly runs into state-space problems while exhaustively verifying

978-3-9819263-2-3/DATE19/©2019 EDAA

timing requirements, the STA-based approach is based on probabilistic symbolic simulations, which can deal very well with a huge state-space.

The main contribution of this paper is the comparison of two state-based SDF timing analysis techniques for multi-processor platform with shared memory communication, namely classical TA model-checking against stochastic symbolic simulation. We demonstrate and evaluate our approach on a Sobel filter by executing the whole flow from SDF application modeling, mapping, scheduling, characterization of the timing behavior to static TA and stochastic TA modeling with an end-to-end timing analysis and comparison of the analysis results.

In the following Sec II we discuss related work and introduce our SDF system and timing model in Sec III. Sec IV describes the characterization of the timing behavior of our SDF application using repeated measurements and statistical inference techniques to obtain representative probability density functions for building an STA model. We compare our statistical analysis results against the static analytical approach in [1] and plain measured data from our FPGA platform (arrow ③). The paper closes with a conclusion and future work.

II. Background & 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.

Different simulation-based approaches have been proposed to support evaluation of multi-core architecture performance early in the design process [8]. Simulation-based approaches require extensive architecture analysis under various possible working scenarios. Created architecture models cannot be exhaustively simulated and worst-case working scenarios are hardly identified and assessed. 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 analyse multi-core systems and provide hard real-time and performance bounds. These formal approaches are commonly classified as state-based approaches and analytical approaches. Since analytical methods depend on solving closed-form equations (characterizing the system temporal behaviour), they have the advantage of being scalable to analyse large-scale 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 [9]. In addition, complex properties such as reachability of certain states cannot be verified by such methods.

State-based approaches consider representation of the architecture under analysis as a transition system. Since the real operation states of the architecture behaviour are reflected, tighter results can be obtained compared to analytical methods. The two main considered application classes are streaming applications (modelled as SDFGs) [4], [5], [10] and generic RT task-based applications [11]–[13]. However, state-based approaches allow exhaustive analysis of system properties at the expense of time-consuming modelling and analysis effort. As illustrated in [1], these approaches can hardly address hardware/software architectures made of high number of heterogeneous components (i.e., application functions, processing resources, memories, communication bus).

Our work considers an intermediate hybrid analysis approach that can be seen as a trade-off between simulation and state-based verification approaches. This approach uses a combination of probabilistic models and analysis techniques. In the context of embedded systems, probabilistic models represent a means of capturing system variability. In this context, variability mainly comes 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 labelled transition system and allow variations about execution times and state transitions to be considered. Analysis of probabilistic models allow quantitative measures to be obtained. Probabilistic approaches that combine simulation and formal approaches appear to be good compromise to deliver both accuracy and limited exploration effort.

Authors in [14], [15] 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 an Stochastic Model Checking (SMC) based analysis capturing the system modus operandi. This enables obtaining more tight values compared to the EVT approach. Yet our method could benefit from their measurement methodology.

An iterative probabilistic approach have been presented by Kumar [3] 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.

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. Various probabilistic model-checkers (some to mention: Plasma-Lab [16], UPPAAL-SMC [17]) support statistical model-checking. SMC has been adopted in [18], [19] to evaluate a many-core architecture based on a 64 processor cores homogeneous platform. A very recent work in [20] demonstrates the applicability of statistical model checking for a quantitative evaluation of uncertainty-aware hybrid AADL designs against various performance queries.

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

III. The System Model

We capture the application in an SDF model [2]. Our overall system model is based on the model of [1], which considers independent processing elements (PEs) and shared communication resources. We extend this model with probability density functions that express the execution and communication time variances.

We introduce our system mode, by using a Sobel filter as a running example.

A. Model of Computation

Our application behavior is captured by an SDF Graph. An example is shown in Fig. 2a. SDFG models offer a strict separation of communication and computation. The SDFG consists of actors, which do the computation part. The directed edges are FIFO-like communication channels. Over those channels, tokens are exchanged between actors. An actor can only be executed, when there are enough tokens available on its ingoing channels. The execution of an actor can be split into three phases: The *read phase*, where tokens get consumed from the channels. The *computation phase*, where those tokens get processed (in this phase, no communication is initiated by this actor and thus no interfere with any other actor occurs during this phase). And the *write phase*, where the actor writes tokens in an outgoing channel. In our example, the communication is based on shared memory access. The repetition vector of an SDFG defines how often each actors must be executed to obtain the initial state of the SDFG again. We call this an iteration. The time that passes from the start of an iteration to its end is called end-to-end latency.

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. This definition is simplified for our example and does not take initial tokens into account.

The Sobel filter example follows SDF semantics and is shown in Fig. 2a. This application consists of four actors, *GetPixel*, *GX*, *GY* and *ABS*. The *GetPixel* actor reads a 9×9 pixel segment from an image. Depending on the position of the pixel in the image, the segment gets extrapolated with zeros. This leads to high variation in execution times. The *GX* and *GY* actors only perform a convolution of the segment provided by *GetPixel*. While the code has always the same execution path, only dynamic behavior of the hardware (e.g. pipeline or cache effects) influences the execution time of those actors. The *ABS* actor calculates the output pixel from the results of the convolutions. Its behavior depend on the sign of the incoming numbers. There are three possible execution paths.

The data transferred via channels is split into tokens. In our example, one token has a size of $4\,\mathrm{Bytes}$ and represents one Pixel. The channels from *GetPixel* to the convolution actors have a buffer size of 81 to transfer the 9×9 pixel segment. The outgoing channels that transfer the results to the *ABS* actor have a buffer size of $4\,\mathrm{Bytes}$ ($1\,\mathrm{Token}$).

B. Model of Architecture

Our hardware model allows the execution of code on multiple processing elements without any interference, as long as they do not explicitly access shared resources (the bus or the shared memory, in our case). Therefore, the code and data of software is places in private memories that can only be accessed by a single processing element (PE). This allows a composable system that work independent from the amount of processing elements. Data exchange between PEs must be done via shared memory.

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 $EP = (\mathcal{T}, \mathcal{M}_s, I)$ 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, and a shared interconnect I.

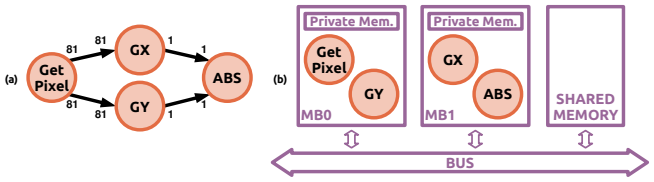


Fig. 2. a) A Sobel filter modeled as an SDFG. A segment of an image gets read within *GetPixel* actor and is sent to the *GX* and *GY* actor. The result of the convolutions in GX and GY is sent to the *ABS* actor where a pixel for the resulting image gets calculated.
b) Our platform consisting of two tiles and a shared memory. The Sobel filter's actors are mapped to the tiles. The instructions and local data of the software uses the private memory, and the channels for data exchange between the actors are mapped to the shared memory.

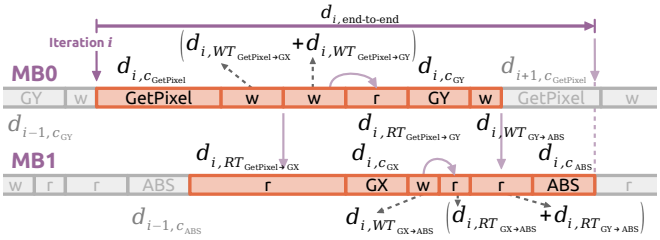


Fig. 3. Expected actor and phase activation for one iteration i (orange - start of the first actor to the end of the last actor) The annotated delays $d_{i,x} \in \vec{D}$ are our measured values for this iteration. $d_{i,RT}$ represent equ. 2 and $d_{i,WT}$ equ. 1. $d_{i,c}$ is the computation time an actor needs. The End-To-End latency $d_{i,\text{end-to-end}}$ is the delay of one iteration.

As shown in Fig. 2b, our platform consists of two MicroBlaze processors with local data and instruction memories, all running at $100\,\mathrm{MHz}$. For inter-processor communication, these processing elements are connected via an AXI interconnect to a shared memory. This memory access is done via single beat First Come First Serve (FCFS) arbitration.

C. Mapping and Scheduling

During the synthesis process the application model is mapped onto the platform model. Every actor of the SDF model is assigned to a tile. Multiple actors on one tile are possible and will be statically scheduled. All channels are mapped onto the shared memory. The private memory is for code and local variables only.

For our example, the *GetPixel* and *GY* actors are mapped onto the first PE. The other two actors, *GX* and *ABS* are mapped onto the other PE, as shown in figure Fig. 2b.

D. 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 take place after the mapping and scheduling process as shown in Fig. 3. 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. The FIFO implementation is optimized for the usage in the SDF context. It is also assumed that all tokens of an actor's port get read or written into the

FIFO at once. This allows a minimal implementation and simplifies our model significantly, since we don't need to care about multiple read or write attempts. Communication between actors can then be modeled considering every single memory access, including those for 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:

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

The time for reading t tokens to the FIFO is:

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

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. The amount n of polling iterations is determined during the analysis of the model and varies between each SDF execution iteration i. When a buffer is ready to be accessed, tokens can be transferred. This consumes read time $d_r \in \vec{D}_r$ or write time $d_w \in \vec{D}_w$ for each token t that is read/written. The memory access time ($\vec{D}_r$, $\vec{D}_w$) include the increased access times due arbitration conflicts on the bus. Beside transferring data, the state of the FIFO buffers must be updated. This state update will also cause read and write accesses. 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. All delays may 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 $A \in \mathcal{A}$.
- $\vec{D_e}, \vec{D_d} \in \mathcal{D}$: 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 from 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.
- d_p: A polling delay when an actor $A_1 \in \mathcal{A}$ tries to access a FIFO that is not ready. This time is defined by the developer and does not change during runtime.

The delay vectors $\vec{D}$ represent all gathered values from the timing measurement. The lowest and highest measured value of each $\vec{D}$ will be used for a BCET/WCET analysis using classical model checking. For the stochastic model checking, a probabilistic density function will be derived from the histogram of the delay vector of each individual part of the SDF application.

E. Timed Automata Model

The system model for the timing analysis consists of four Timed Automata templates: *Tile0*, *Tile1*, *Com* and *Bus*. An overview showing the interaction between those TA template is depicted in Fig. 4. One template is used for each tile $T \in \mathcal{T}$ (*Tile0*, *Tile1*), modeling timing behavior of mapped and scheduled SDF actors $A \in \mathcal{A}$ on each tile. The *Com* template models the timing behavior of actors in the read/write phases from/to a FIFO channel $C \in \mathcal{C}$ and the communication over the Bus. There is a separate instance of *Com* for each tile. The

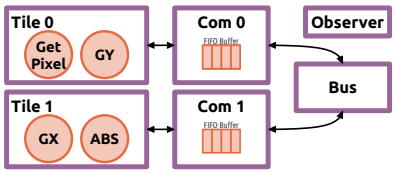


Fig. 4. Overview of the TA model. Two TAs modeling the mapped and scheduled SDF application on their tiles. Each tile has an instance of a communication TA template. The Bus TA models the contention on the bus and the memory access time.

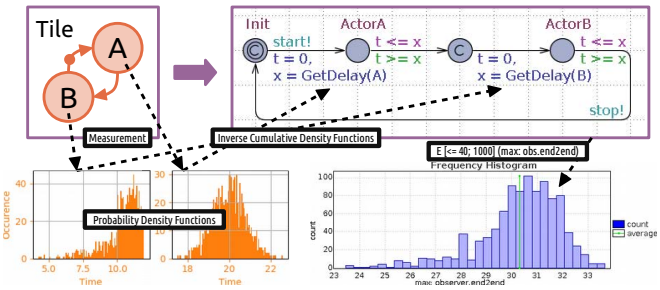


Fig. 5. Example of a stochastic Timed Automata modeling a two-actor SDF on a single tile. The measured timing characteristic can be seen in the orange histograms below the tile. The blue histogram is the result of the end-to-end latency analysis with UPPAAL SMC.

fourth TA template *Bus* models the read and write accesses to the Bus and implicitly to the shared memory.

In this paper, we use this network of TA templates for two purposes. One for classic model checking, as described in [1] and one for stochastic model checking. The classic MC timing analysis is based on the WCET and BCET intervals of the measured data, while the SMC model uses a PDF to model the execution time of the actors. Both models tailored to our for-running example (Sobel filter on 2-tiles platform), without loss of generality, are provided for a detailed introspection under *https://gitlab.uni-oldenburg.de/stemmer/date19*, since showing them here is due to space limitations not possible.

Instead, a very simple example of how a stochastic TA model can be annotated with measured delays is shown in Fig. 5. In this example, two actors on a single tile are shown where communication is not considered. The orange histograms (left) represent the measured values of the execution time of the single actors. For analysis, the modeled delay distribution of the actors should be as close as possible to the measured distribution. To reach this goal, a PDF gets derived from the measured data. The only random number generation provided by UPPAAL generates uniform distributed floating point numbers. The delay distribution of actor *A* could be approximated with a Box-Muller-Transformation [21] because of the normal distributed characteristic. Actor *B* requires a more sophisticated approximation. Therefore a discrete inverse cumulative distribution function (ICDF) gets derived from the PDF of the measured timings. Then the uniform random number generator can be used to select delays from the ICDF with a distribution close to the measured delays. In the TA in Fig. 5, the function *GetDelay* returns a random delay following the modeled distribution. The blue histogram on the right shows the resulting distribution of the analyzed end-to-end latency of the SDF application. It is clearly to see how actor *B* influences the average execution time.

IV. Evaluation

For evaluation we used a Sobel filter running on a 2-tiles platform (see Fig. 2). We used the measuring method introduced in [22], which allows us to measure the delay of desired parts of the filter to annotate the timing model presented in Sec III. To evaluate our approach (Fig. 1 ①) we compared our SMC based timing analysis with the classical model checking approach in [1] using the same data set ②. We also compare the results to the measured best-case and worst-case execution times for the whole graph ③.

A. Setup of The Experiment

For our experiments we implemented the Sobel filter as shown in Fig. 2a. The Sobel filter is an edge detection filter and was implemented to follow the SDF semantics. With this filter, we process a 48×48 pixels image.

The actor *GetPixel* reads a 9×9 matrix around the current pixel to process from the memory where the image's raw data was stored at. The execution time of *GetPixel* (Fig. 6a) is highly varying due to the extrapolation of the matrix at the edges of the image. The matrix then gets communicated to the *GX* and *GY* actors. Those actors do a convolution with a matrix of constants. Their execution delays depend on the data at computation phase (Fig. 6b and 6c) and the scheduling at read phase. The data dependency is caused by the multiplication. The compiler adds a check for operands being 0. *ABS* has a branched behavior, but the operations are simple which leads to three possible delays at the computation phase (Fig. 6d).

Our evaluation platform is the *Xilinx ZC702 Evaluation Board* with a *Zynq XC7Z020*. For the experiments we only use the FPGA part of the Zynq. The development tool chain is *Xilinx Vivado 2016.3* improved with a *GCC 6.3.0* compiler, *binutils 2.27* and *newlib 2.5.0* As seen in Fig. 2b all actors are mapped on *MicroBlazes (MB)*. The MBs are configured to have no caches and all local variables and instructions are mapped to their private memory. The actors are scheduled with *static order scheduling*. All communication channels are mapped to the shared memory. Our bus protocol is *First come First served* with a *Single Beat* characteristic.

B. Timing Measuring

In our evaluation all timings are given in cycles since this unit is independent from the setup of the system clock. In order to get the worst case scenario for our edge detection algorithm we used an 8bit gray scale noise picture (48×48 pixels) generated in *GIMP 2.8* with RGB noise generator.

The read and write delay for the shared memory access only provides two different times, depending whether the bus is busy or not. For communication, the *enqueue*, *dequeue* and *pollingdelay* time is independent from the SDF channel and actors. The measured times for communication and the BCET/WCET for computation are shown in Tab. I. For the computation time of an actor we used the inverse cumulative distribution functions shown in Fig. 6.

C. Results

In Tab. II the evaluation results are presented. All delays represent end-to-end latencies of the analyzed Sobel filter. The *Measured data* row shows the actual measured best-case, worst-case and average-case end-to-end latency observed during 1 Mill. measurements. The *Classic MC* row shows the result of the state-based analysis using only the WCET and BCET shown in Tab. I. The *Stochastic MC* rows show the results of the SMC approach using different PDFs of the measured timings.

TABLE I
THE MEASURED BEST-CASE AND WORST-CASE DELAYS FOR OUR MODELS

Delay	Best Case	Worst Case
$\vec{D}_e$ (enqueue)	181 Cycles	185 Cycles
$\vec{D}_d$ (dequeue)	142 Cycles	144 Cycles
$\vec{D}_r$ (read)	18 Cycles	19 Cycles
$\vec{D}_w$ (write)	14 Cycles	15 Cycles
$\vec{D}_p$ (polling)	251 Cycles	251 Cycles
$\vec{D}_{c_{GetPixel}}$	7811 Cycles	8245 Cycles
$\vec{D}_{c_{GX}}$	6563 Cycles	6779 Cycles
$\vec{D}_{c_{GY}}$	6566 Cycles	6776 Cycles
$\vec{D}_{c_{ABS}}$	54 Cycles	72 Cycles
$\vec{D}_{iter}$ (end-to-end)	38 170 Cycles	38 806 Cycles

TABLE II
COMPARISON OF THE ANALYZED END-TO-END LATENCIES (IN CYCLES) WITH THE REAL SYSTEM AND THE ESTIMATED PROBABILITY TO FINISH THE ITERATION BEFORE A DEADLINE OF 40 000 Cycles IS REACHED. THE SMC RESULTS HAVE A CONFIDENCE OF 99.95 %.

Subject	BC	WC	Mean	In Time
Measured data	38 170	38 806	38 453	-
Classic MC	38 171	45 208	41 690	-
Stochastic MC (unif.)	38 617	44 885	41 914	20.3 %
Stochastic MC (normal)	38 719	44 744	40 671	38.1 %
Stochastic MC (ICDF)	38 769	44 659	40 396	76.2 %

The analyzed mean execution time is 1.14 times closer to the actual mean execution time when using the measured distribution instead of assuming a normal distribution, and 1.67 times closer than classic model checking.

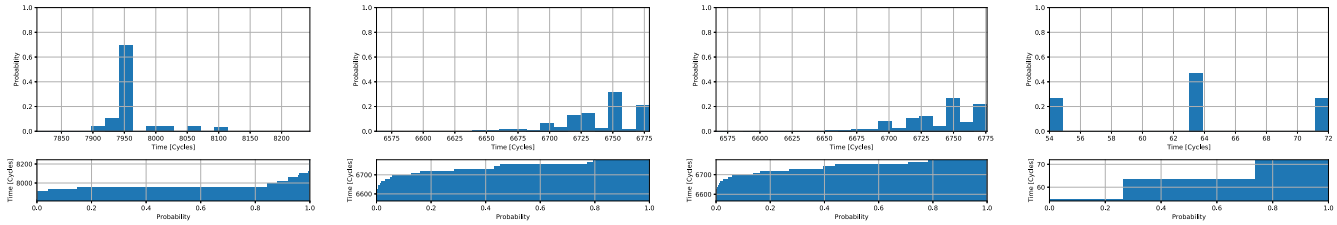
The results are also plotted in Fig. 7. In this figure, the median is shown as dashed line. The median of the ICDF has an offset of 4.9 % compared to the real measured values. The normal distribution is 5.6 % off, the uniform 8.9 %. This shows the significance of the PDF.

For the classic TA analysis we used the UPPAAL query `inf{obs.Finish}: obs.end2end` for the BCET analysis, and `sup{obs.Finish}: obs.end2end` for WCET. For the stochastic analysis we used the UPPAAL SMC query `Pr [<= 50000](<> obs.Finish)` with UPPAAL's *statistical parameters* set to $\alpha, \beta, \epsilon = 0.005$ for a confidence of 99.95 %.

One advantage of SMC is the possibility to check how likely it is to not miss a certain deadline (here 40 000 Cycles) by applying the query `Pr [<= 50000](<> obs.Finish && obs.end2end <= 40000)`. Tab. II, 4th column shows how likely an iteration is finished before the deadline is reached. Our ICDF based model predicts a probability of 76.2 % with a certainty of 99.95 %. Other PDFs are more pessimistic.

Since the BCET of *GX* and *GY* is relative unlikely as you can see in Fig. 6b,c (less than 0.05 % for the first 3 bins) the results of the SMC is less close to the measured BCET than the classic MC. Furthermore, we used a histogram with only 20 bins to get a discrete ICDF that can be manually annotated to the UPPAAL model. All results have a small offset to the measured delays due to the selected bin size as well as a pessimistic communication model.

The analysis time highly depends on the desired confidence. The Classic MC took 34.35 s. For a confidence of 99.5 % the SMC analysis took only 13.54 s while 99.95 % took 214.6 s.



(a) GetPixel (b) GX (c) GY (d) ABS

Fig. 6. Histogram of measured computation phase timings of the actors and the inverse cumulative distribution functions for each actor used in the model.

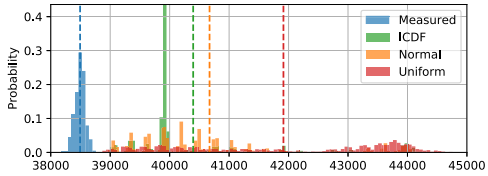


Fig. 7. Histograms of the end-to-end latency analysis with different PDFs compared with the real measured values. The dotted line are the median.

V. Conclusion & Future Work

In this work, we have presented a probabilistic state-based RT analysis approach, which uses real measured execution times to analyze the timing behavior of SDF applications with the help of statistical model-checking. The viability of our approach (Fig. 1) was demonstrated on a Sobel filter running on a 2 tiles platform implemented on top of a Xilinx Zynq 7020. While classic MC only uses a BCET and WCET, the SMC approach allows a more sophisticated model of the execution time distribution. The bin size of the histogram used to derive the PDF is crucial for the quality of analysis results. Furthermore, the used statistical inference process has a huge impact. In future work we plan to use the Kernel Density Estimation technique [23] that will be applied to estimation a more accurate PDF from measured data. We will also investigate in the statistical dependencies between the actors. To further improve our models, we will use Plasma-Lab [16] in combination with a SystemC model instead of UPPAAL-SMC. This allows adding functional behavior to our model as well as using external libraries for more complex PDFs.

References

[1] 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.

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

[3] A. Kumar, "Analysis, design and management of multimedia multi-processor systems," Ph.D. dissertation, Eindhoven University of Technology, 2009.

[4] 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.

[5] 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 *International Symposium on Formal Methods*. Springer, 2015, pp. 551–569.

[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] J. Bengtsson and W. Yi, "Timed Automata: Semantics, algorithms and tools," in *Advanced Course on Petri Nets*. Springer, 2003, pp. 87–124.

[8] A. Gerstlauer, C. Haubelt, A. D. Pimentel, T. P. 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, Oct 2009.

[9] 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

[10] 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

[11] C. Norstrom, A. Wall, and W. Yi, "Timed automata as task models for event-driven systems," in *Proceedings Sixth International Conference on Real-Time Computing Systems and Applications. RTCSA'99 (Cat. No.PR00306)*, Dec 1999, pp. 182–189.

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

[13] 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*, Nov 2010, pp. 339–349.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] Y. Bao, M. Chen, Q. Zhu, T. Wei, T. Zhou, and F. Mallet, "Quantitative Performance Evaluation of Uncertainty-Aware Hybrid AADL Designs Using Statistical Model Checking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 12, pp. 1989 – 2002, Dec. 2017. [Online]. Available: https://hal.inria.fr/hal-01644285

[21] G. E. P. Box and M. E. Muller, "A note on the generation of random normal deviates," *Ann. Math. Statist.*, vol. 29, no. 2, pp. 610–611, 06 1958. [Online]. Available: https://doi.org/10.1214/aoms/1177706645

[22] 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.

[23] 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