Latency Analysis of Homogeneous Synchronous Dataflow Graphs Using Timed Automata

Guus Kuiper*
g.kuiper@utwente.nl

Marco J.G. Bekooij*†
marco.bekooij@nxp.com

*University of Twente, Enschede, The Netherlands

†NXP Semiconductors, Eindhoven, The Netherlands

***Abstract*—There are several analysis models and corresponding temporal analysis techniques for checking whether applications executed on multiprocessor systems meet their real-time constraints. However, currently there does not exist an exact end-to-end latency analysis technique for Homogeneous Synchronous Dataflow with auto-concurrency (HSDF^a) models that takes the correlation between the firing durations of different firings into account.**

In this paper we present a transformation of strongly connected HSDF^a models into timed automata models. This enables an exact end-to-end latency analysis because the correlation between the firing durations of different firings is taken into account.

In a case study we compare the latency obtained using timed automata and a Linear Program (LP) based analysis technique that relies on a deterministic abstraction and compare their run-times as well. Exact end-to-end latency analysis results are obtained using timed automata, whereas this is not possible using deterministic timed-dataflow models.

I. Introduction

Cyber-Physical Systems (CPSs), such as braking systems of cars are safety-critical. These systems are implemented using multiple processors that execute applications that must satisfy real-time constraints. Satisfaction of these real-time constraints must be verified using formal analysis techniques.

Timed-dataflow models are suitable for modeling certain types of CPSs. Analysis techniques have been developed for these types of models to determine the guaranteed throughput and maximum latency [13]. The behavior of some of these timed-dataflow models can be described using Max-Plus algebra [5]. Approximation techniques are often applied in order to reduce the computational complexity of the throughput and latency analysis methods of these models [9]. These approximation techniques make use of a deterministic abstraction which reduces the accuracy.

Another approach is to model these real-time systems with timed automata and to use a model checker like Uppaal [4] to compute the exact maximum end-to-end latency of a system, which is determined by considering all possible cases. However, such an exhaustive analysis might increase the run-time of the analysis significantly.

The HSDF^a model is closely related to the Homogeneous Synchronous Dataflow (HSDF) model. Like in the HSDF model, the actors in the HSDF^a model produce one token on each output queue and consume one token from each input queue per firing. However, unlike the HSDF model, the dependencies between firings are preserved despite that actors have a varying firing duration and are executed auto-concurrently. Auto-concurrency can be used to model data parallel executions of tasks [8]. As a result, the behavior of the HSDF^a model can be described with Max-Plus algebra, whereas this is not possible for the HSDF model. Furthermore, the correlation between the firing durations of actor firings can be expressed. However, currently no exact end-to-end latency analysis technique exists that takes this correlation into account.

In this paper we present a transformation of strongly connected HSDF^a models into timed automata. We show that by transforming HSDF^a models into times automata, we can compute the exact end-to-end latency using Uppaal. In the case study we compare the latency for two HSDF^a models computed using a timed automata approach with the latency obtained using an LP based analysis technique that relies on a deterministic abstraction of the HSDF^a model. We also compare the run-times of these approaches.

II. Related work

In this section we present work that is related to the modeling and analysis of dataflow graphs. We first discuss the relation between some dataflow models and the HSDF^a model. Then we discuss work related to the transformation of time Petri nets (TPNs), to which the HSDF^a model is closely related, into timed automata. Finally, we discuss related work on other end-to-end latency analysis techniques for dataflow models.

The Synchronous Dataflow (SDF) model supports integer production and consumption rates and is a generalization of the HSDF model. The Cyclo-Static Dataflow (CSDF) model [3] has phases and is a generalization of the SDF model. Each phase can have a different consumption and production rate and these phases fire in a fixed cyclic order. In the HSDF, SDF and CSDF models, tokens can overtake each other as a result of auto-concurrency. In the Cyclo-Static Dataflow with auto-concurrency (CSDF^a) model [11], which is based on CSDF, an index is added to tokens to take care that dependencies between firings are independent of the production order of tokens. We consider the variant of the CSDF^a model with one phase and single rate and refer to it as HSDF^a.

A transformation of TPNs into timed automata to determine the end-to-end latency is presented in [6]. Both HSDF^a graphs and Petri nets do allow reordering of tokens, but a key difference is that Petri nets do not preserve dependencies between iterations of data-dependent actors. As a result, the

978-3-9815370-8-6/17/$31.00

temporal behavior of HSDF^a graphs can be described with Max-Plus algebra, which does not hold in general for Petri nets. Petri nets support auto-concurrency. However, we show that auto-concurrency cannot be modeled in a timed automaton in general, which contradicts the claim made in [6] that the transformation is always possible.

End-to-end latency analysis techniques for self-timed executed SDF graphs are presented in [13]. The approach in [13] does consider bursts of events. However, the approach does not take into account that firing durations of actors can be correlated. This is also the case for the timed automata based throughput analysis approach in [1]. Our approach does allow the firing durations of actors to vary each firing and takes the correlation between the firing durations of different firings into account. Our approach also supports auto-concurrent execution of actors but requires the HSDF^a graphs to be strongly connected.

III. The HSDF^a model

In this section we first present the HSDF^a model. Then we give a definition of the maximum end-to-end latency for this model.

An HSDF^a graph is a directed graph $G = (V, E, \delta, \rho)$ that consists of a set of actors V connected by a set of directed edges E. An actor $v_i \in V$ communicates with another actor v_j by producing tokens on an edge $e_{ij} \in E$. Each edge represents an unbounded buffer instead of a queue because tokens can be written into these buffers in a different order than they are read. Initially there are $\delta_{ij} \in \mathbb{N}_0$ tokens on an edge. An arrival of a token is an event which is represented by the tuple (ϑ, ι, τ) consisting of a value ϑ, an index ι, and a time-stamp $\tau \in \mathbb{R}_{\geq}0$. An HSDF actor v_i is enabled to fire its i-th iteration if there is at least one token with index i on all its incoming edges. During self-timed execution of an HSDF^a graph, each actor fires immediately after it is enabled. When an actor fires, one token is consumed from each of its incoming edges and one token with index i is produced on each outgoing edge of the actor. The tokens are produced exactly $\rho_x(i) \in [\check{\rho}_x, \hat{\rho}_x]$, chosen non-deterministically, after the enabling of the actor. The firing duration $\rho_x(i)$ of the i-th firing of an actor v_x can be defined more precisely using e.g. a non-deterministic finite state-machine. Multiple firings of an actor overlap if there are sufficient input tokens with the required indices. A token with index $i+1$ can be produced by an actor before the token with index i is produced even if input token i and $i+1$ arrive at the same point in time. This is because firing $i+1$ can have a smaller firing duration than firing i. However, the consumption order of tokens is determined by the indices and is therefore independent from the production order.

We define the end-to-end latency as $L_{sd} = \max_i \big(d(i) - s(i)\big)$ with $s(i)$ the arrival moment of the token with index i in the input buffer, and $d(i)$ the production moment of the token with index i in the output buffer. Figure 1 shows a basic HSDF^a example consisting of two actors,v_x and v_y, for which $L_{sd} = \rho_x(i) + \rho_y(i)$ in case there are always sufficient tokens on e_{yx}. However, a burst of events at the input can, for example, cause a lack of sufficient tokens on e_{yx}. In that case v_x is enabled later, and as a result, the maximum end-to-end latency is increased.

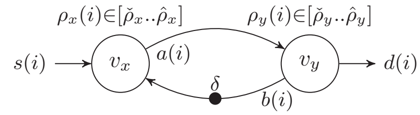


Fig. 1: HSDF^a graph containing a cycle of edges including δ tokens.

IV. Max-Plus Semantics of HSDF^a

In this section Max-Plus equations are presented that describe the semantics of the HSDF^a model.

Max-Plus algebra [2] only uses a max operator $\oplus$, and a plus operator $\otimes$. The $\otimes$ operator has a higher precedence than the $\oplus$ operator. The production events of actors during self-timed execution can be described using these operators.

The maximum end-to-end latency for a dataflow graph can be described by combining expressions for actors and edges. Tokens on an edge represent dependencies of the consuming actor on a firing with a lower index of the producing actor. Waiting until an actor is enabled by all its inputs is enforced by the $\oplus$ operator. The delay caused by the firing duration is added in Max-Plus algebra using the $\otimes$ operator. We can now derive L_{sd} for the cyclic HSDF^a graph as shown in Figure 1 as follows:

$$a(i) = \big(s(i) \oplus b(i-\delta)\big) \otimes \rho_x(i) \quad (1)$$

$$d(i) = b(i) = a(i) \otimes \rho_y(i) \quad (2)$$

$$L_{sd} = \big(s(i) \oplus b(i-\delta)\big) \otimes \rho_x(i) \otimes \rho_y(i) \quad (3)$$

In these expressions $a(i)$ and $b(i)$ represent the moments at which the token with index i is produced on e_{xy} and e_{yx} respectively.

V. Timed automata model of HSDF^a graphs

In this section we describe the derivation of an extended timed automaton that is semantically equivalent to an HSDF^a graph. A complete HSDF^a graph is composed as a network of timed automata.

A. Dataflow edge model

Global variables are used in UPPAAL to represent the number of tokens on the edges of a strongly connected HSDF^a model. Additional information about the tokens on an edge, such as their indexes, is stored in a list. Together these form queues which can be manipulate using access functions. A token is produced using the `produce()` function and consumed using the `consume()` function. The `empty()` function returns if the queue is empty. The number of initial tokens on an edge determines the initial state of the queue.

B. Actor model

A timed automaton for a dataflow actor can now be composed based on the above described Max-Plus equations. The template for an actor that does not fire auto-concurrently is shown in Figure 2. Two locations and one clock are used: One to represent a state in which the actor is waiting until sufficient tokens with the required index are present on all incoming edges, and one in which the actor is executing. The waiting location is left immediately when all incoming edges contain

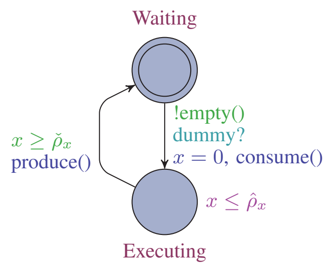


Fig. 2: Automaton modeling an actor that cannot fire autoconcurrently.

tokens, which is ensured by means of the guard function `empty()` and the urgent channel `dummy`. Tokens are removed from the input edges using the `consume()` function. In the executing location the firing duration of the actor is modeled using a combination of an *invariant* and *guard*. Finally, after the firing duration, a token is produced on all corresponding outgoing dataflow edges using the function `produce()`.

Auto-concurrency can only be modeled in UPPAAL in case an upper bound N on the number of actor replications that can execute concurrently is known. In that case, we replicate the automaton in Figure 2 N times in the UPPAAL model. Each replication has a local variable to keep track of the index of the next token that it should consume. Fortunately, in strongly connected HSDF^a graphs, there are per definition never more tokens on a cycle than the number of initial tokens. As a consequence, also the number of replicas of an actor that can fire concurrently is never larger than the minimum number of initial tokens on the cycles to which the actor belongs.

C. Complete automaton of an HSDF^a graph

A complete timed automaton can now be defined in UPPAAL for an HSDF^a graph by instantiating components in the automaton for each element of the dataflow graph. For each actor v_i the actor template is instantiated. The edges e_{ij} in the dataflow graph are modeled using global variables and access functions.

To model a source the timed automata model as described in [10] is used. This source can generate bursts of events. The source is defined by its period P and maximum jitter J. The jitter J can be larger than P. A similar automaton, as described in [10], is used for measuring the end-to-end latency.

VI. CASE STUDY

In this section we compare latency analysis using an UPPAAL model with an LP-based dataflow analysis technique. We consider dataflow models in which each actor contains an internal non-deterministic finite state machine which determines the maximum firing duration of the actor in each iteration. These firing durations are not always equal to the worst-case firing durations of the actor and therefore the workload of each actor varies. In [7] an approach to capture a varying workload in a deterministic dataflow model is described. This enables the use of computationally efficient latency and throughput analysis techniques. In this case study, we use the analysis method presented in [12] which makes use of LPs. We also analyze the latency of a WLAN 802.11p receiver application.

TABLE I: Latencies obtained using UPPAAL and using a deterministic HSDF^a model for two values of δ_{yx}.

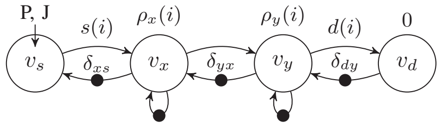
		$\delta_{yx} = 1$			$\delta_{yx} = 2$		
P (ms)	J (ms)	run-time (s)	L_{sd} (ms)	L_{sd} [7] (ms)	L_{sd} (ms)	L_{sd} [7] (ms)	L_{sd} (ms)
4	0	0.02	4	4	4	4	4
4	4	0.15	6	8	5	5.5	6
4	8	0.47	10	12	8	8	8
4	12	1.24	12	16	9	9.5	10
4	16	3.06	16	20	12	12	12

The HSDF^a graph that we first consider in this section is shown in Figure 3a and consists of the actors v_x and v_y, a source v_s, and a destination v_d. The source is characterized by its period P of 4 ms and jitter $J \in \{0, 4, 8, 16\}$ ms, which allows for bursts of tokens to arrive on $e_{sx.}$ The maximum number of tokens that can arrive at the same time on e_{sx} follows from these parameters and equals $\lfloor J/P \rfloor + 1$. Both actors have a firing duration which depends on the internal state of the actors. In this example we consider actors with two internal states, which results in a firing duration of 1 ms in one state, and a firing duration of 2 ms in the other state. It is possible to remain for a number of iterations in the first state, but the second state is always switched to the first state immediately. It is therefore not possible to have two consecutive firings with a firing duration of 2 ms. Such an automaton models for example the sporadic execution of an additional code-segment inside a task. We guarantee that there are sufficient initial tokens δ_{xs} and δ_{dy} such that v_s and v_y are never delayed by the lack of tokens on e_{xs} or e_{dy}. We want to analyze the maximum latency L_{sd} from the source to the destination of this HSDF^a graph.

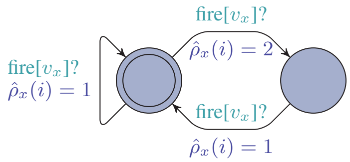
A more accurate workload characterization can be created for actor v_x than non-deterministically selected firing durations in an interval, by making use of the additional information about the state machine inside the actors that determines the firing duration. Such a method is described in [7], which requires that two parameters are determined: a guaranteed throughput ρ and latency σ. For both actors v_x and v_y the minimum throughput is obtained when switching happens continuously between the two internal states. This results in a guaranteed throughput of two tokens per 3 ms, therefore, $\rho = 1.5\,\text{ms}$. The parameter σ in the model describes the maximum latency and is equal to 2 ms. Using these two parameters, a more accurate dataflow model can be created for v_x using two actors $v_{x,0}$ and $v_{x,1}$ as shown in Figure 3c. One actor has a self-edge with a single token and a firing duration equal to ρ_x, whereas the other actor has a firing duration of $\sigma_x - \rho_x$.

The additional information about internal states can also be incorporated in a timed automaton. Figure 3b shows an automaton which determines $\rho_x(i)$. Every time v_x fires the state is updated based on the urgent channel `fire`. This channel replaces the channel `dummy` in the actor model of Figure 2. The value of $\hat{\rho}_x$ in the actor model is made equal to the firing duration $\hat{\rho}_x(i)$ as indicated by the internal state machine. The value of $\check{\rho}_x$ is set to 0.

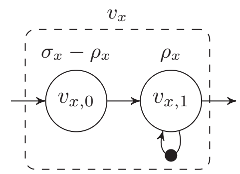
The computed latencies for different jitters are presented in Table I. The results obtained using timed automata are in the fourth and sixth column. The results obtained using the two parameter dataflow model are in the fifth and seventh column.



(a) HSDF^a model with $\rho_x(i)$ and $\rho_y(i)$ determined by an internal state machine.



(b) Timed automaton of the internal state machine of v_x which determines its firing duration.



(c) Two parameter dataflow model of v_x.

Fig. 3: HSDF^a model used in the case study, internal state machine of the actors, and deterministic 2 parameter workload model for HSDF^a actor v_x.

The table shows that more accurate result are obtained using timed automata for $\delta_{yx} = 1$, for a jitter of 4 and 12 ms and equal results for the other jitter values. Adding an additional token on e_{yx} improves the result of the LP based dataflow analysis method. All results were computed within $1\,\mathrm{ms}$ using this method. However, model-checking still results in more accurate analysis results. Furthermore, it can be seen that the run-time of the model checker is rapidly increasing for larger jitter values despite that this example considers a very small problem instance. The negligible run-time and its better scaling of the LP based dataflow analysis method makes it more suitable for larger graphs.

The last column of Table I contains the results obtained for the case that the internal state machine of the actor v_x is ignored and non-deterministically selected firing durations in the interval $[\check{\rho}_x, \hat{\rho}_x]$ are used instead. These results are the most pessimistic ones and are the same for both analysis methods.

An HSDF^a model of a WLAN 802.11p application is presented in [12]. Figure 4 shows this HSDF^a model which consists of a source with a period of $10\,\mu$s, 8 actors and 19 edges connecting them. The number of initial tokens, $\delta_1 \ldots \delta_8$, are all set to 3. An automaton per actor, like the one in Figure 3b, determines their firing durations. UPPAAL is used to calculate L_{sd} which is equal to $21\,\mu$s and is computed in $435\,$s. A latency of $23\,\mu$s was obtained with the LP based dataflow analysis method within $1\,$ms. The run-time of UPPAAL can be reduced at the cost of accuracy by removing some of the automata and instead using actors with non-deterministically selected firing durations from their interval. Replacement is allowed because it results in that more behaviors are considered during analysis. The run-time is $3\,$s and latency $23\,\mu$s in case all actors are replaced.

VII. Conclusion

In this paper we presented a behavior-preserving transformation of strongly connected HSDF^a graphs into timed automata. These timed automata allow for the computation of exact end-to-end latencies because the correlation between the firing durations of different firings is taken into account.

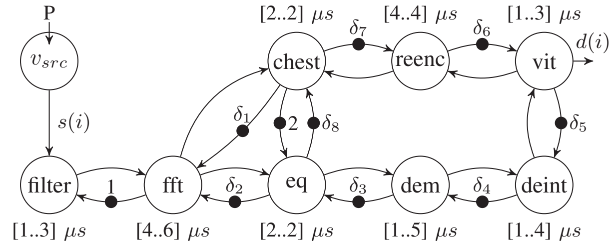


Fig. 4: Dataflow graph of a WLAN 802.11p transceiver application. All actors have an implicit self-edge with a single token.

The transformation of HSDF^a graphs into a behaviorally-equivalent timed automata is possible because the number of tokens on edges in a strongly connected HSDF^a model is bounded. Therefore, buffers can be modeled using the extended timed automata of UPPAAL which by definition have a finite number of states. This also guarantees that there is a maximum number of replicas of the same actor that can fire concurrently. This guarantees that there is a finite number of concurrent state machines, and thus states, required to model each HSDF^a actor.

In the case study we consider two HSDF^a examples for which exact end-to-end latency analysis results are obtained using timed automata and UPPAAL, whereas this is not possible using deterministic timed-dataflow models. This comes at the cost of a higher run-time which for the considered examples is less than $435\,$s.

References

[1] W. Ahmad et al. Resource-constrained optimal scheduling of synchronous dataflow graphs via timed automata. In *ACSD*, pages 72–81, 2014.

[2] F. Baccelli et al. *Synchronization and linearity: an algebra for discrete event systems*. John Wiley & Sons Ltd, 1992.

[3] G. Bilsen et al. Cyclo-static dataflow. In *ICASSP*, volume 5, pages 3255–3258, 1995.

[4] A. David, J. Illum, K. G. Larsen, and A. Skou. Model-based framework for schedulability analysis using UPPAAL 4.1. *Model-based design for embedded systems*, 1(1):93–119, 2009.

[5] R. de Groote et al. Max-plus algebraic throughput analysis of synchronous dataflow graphs. In *SEAA*, pages 29–38. IEEE, 2012.

[6] Z. Gu and K. G. Shin. Analysis of event-driven real-time systems with Time Petri Nets. In *Design and Analysis of Distributed Embedded Systems*, pages 31–40. Springer, 2002.

[7] J. P. H. M. Hausmans et al. Two parameter workload characterization for improved dataflow analysis accuracy. In *RTAS*, pages 117–126, 2013.

[8] J. P. H. M. Hausmans et al. Unified dataflow model for the analysis of data and pipeline parallelism, and buffer sizing. In *MEMOCODE*, pages 12–21. IEEE, 2014.

[9] J. P. H. M. Hausmans et al. A refinement theory for timed-dataflow analysis with support for reordering. In *EMSOFT*, page 20. ACM, 2016.

[10] M. Hendriks and M. Verhoef. Timed automata based analysis of embedded system architectures. In *Proc. Parallel & Distributed Processing Symp.*, pages 8–pp. IEEE, 2006.

[11] P. Koek et al. CSDF^a: A model for exploiting the trade-off between data and pipeline parallelism. In *SCOPES*, pages 30–39. ACM, 2016.

[12] P. S. Kurtin et al. Combining offsets with precedence constraints to improve temporal analysis of cyclic real-time streaming applications. In *RTAS*, pages 1–12, 2016.

[13] O. M. Moreira and M. J. G. Bekooij. Self-timed scheduling analysis for real-time applications. *EURASIP Journal on Advances in Signal Processing*, 2007(1):1–14, 2007.