A Dynamic Computation Method for Fast and Accurate Performance Evaluation of Multi-Core Architectures

Sébastien Le Nours
University of Nantes, UMR CNRS 6164 IETR
Polytech Nantes, La Chantrerie
44306 Nantes, France
sebastien.le-nours@univ-nantes.fr

Adam Postula, Neil W. Bergmann
University of Queensland
School of Information Technology and Electrical Engineering
Brisbane, Australia
adam@itee.uq.edu.au, bergmann@itee.uq.edu.au

***Abstract*—Early estimation of performance has become necessary to facilitate design of complex multi-core architectures. Performance evaluation based on extensive simulations is time consuming and needs to be improved to allow exploration of different architectures in acceptable time. In this paper, we propose a method that improves the tradeoff between simulation speed and accuracy in performance models of architectures. This method computes during model execution some of the synchronization instants involved in architecture evolution. It allows grouping and abstracting architecture processes and this way significantly reduces the number of simulation events. Experiments show significant benefits from the computation method on the simulation time. Especially, a simulation speed-up by a factor of 4 is achieved in the considered case study, with no loss of accuracy about estimation of processing resource usage. The proposed method has potential to support automatic generation of efficient architecture models.**

I. Introduction

The design complexity of embedded systems is directly related to the necessity to integrate in the most efficient manner increasing numbers of heterogeneous components. The process of system architecting aims at containing such complexity by organizing and supervising the various design steps from system specification to components integration and test. Especially, performance and cost of potential architectures have to be assessed early in the design cycle to prevent costly design iterations. Simulation-based performance evaluation approaches allow executable models of architectures to be created and analyzed according to potential working scenarios. However, an open issue related to applying such approaches to complex architectures concerns the existing compromise between achievable simulation speed and accuracy of estimated performance.

In this context, the emergence of the transaction level modeling (TLM) paradigm has allowed platform resources, and so architectures, to be modeled and simulated at a higher abstraction level than traditional register transfer level. This paradigm relies on an explicit separation between communication and computation mechanisms. It also considers an event-driven evolution of models, for which events reflect specific synchronization instants among processes forming the architecture model. A currently well disseminated simulation kernel is the one attached to the SystemC language [1]. It aims at correctly managing the time-ordered sequence of events among architecture processes and the advancement of simulation time. In case of performance evaluation of multiprocessor architectures, synchronizations among processes are especially needed to correctly express access to shared resources of the platform. These are performed by calling the SystemC `wait()` statement, implying time-consuming context switches in the simulation kernel. Different approaches can be adopted to limit the amount of required events during simulation and thus to improve achievable simulation speed. One solution is represented by the *loosely-timed* coding style (TLM-LT) defined in the SystemC TLM-2.0 standard [1]. This coding style supports the temporal decoupling method that allows processes to run ahead in a local time with no use of the simulator. The definition of a global quantum is thus needed to impose an upper limit on the time a process is allowed to run ahead of simulation time. However, too large a value can lead to degraded timing accuracy because delays due to access conflicts to shared resources are not simulated. Specific simulation methods are therefore required to favour creation of efficient models with limited amount of events.

This paper presents a new method that leads to a significant reduction of simulation events and still maintains accurate estimation of platform resources usage. This method uses a formal description of synchronization instants and time dependencies among architecture model processes. Specific instants when usage of platform resources is modified are expressed using this formalism. In this paper, we use the term *evolution instants* to designate these instants. Based on these expressions, evolution instants are dynamically computed during model execution. Besides, a local time is used to observe computed evolution instants with no use of the simulator. Therefore, the proposed method allows some of the architecture processes to be combined into a single equivalent executable model as seen by the simulator, which greatly reduces the number of events and context switches that are handled by the simulation kernel. Compared to related work, the contribution of this paper is about the way formal expression of evolution instants and simulation are combined to improve performance models efficiency. In the scope of this paper, this approach is presented for statically scheduled architectures with no pre-emption. This

978-3-9815370-2-4/DATE14/©2014 EDAA

assumption is commonly considered in the case of dataflow oriented applications for which the time schedule of computations does not vary during execution. However, considered architectures are data-dependent and computation execution times are variable, which makes it difficult to predict usage of resources. We introduce a didactic example to illustrate application of the method. Impact of the computation method on performance models' execution time is then evaluated for more complex situations. Significant improvement of the compromise between simulation speed and accuracy is observed. Benefits of the proposed approach are illustrated through the study of a heterogeneous architecture of a communication receiver implementing the physical layer of the Long Term Evolution (LTE) protocol.

This paper is organized as follows. Related work is explained in Section II. In Section III, we describe the proposed simulation method and the adopted algebraic formalism. Implementation details are provided in Section IV. Benefits of the approach are illustrated in Section V through a case study. Finally, conclusions are drawn in Section VI.

II. Related work

The main features of simulation-based performance evaluation approaches are discussed in [2]. Performance models of architectures are typically formed by combination of application and platform models. For the purpose of performance evaluation, an application is often modeled without considering a complete description of functionalities. Workload models are thus used to express computation and communication loads that an application causes when executed. During application execution, architecture evolution comes from interpretation of such loads by the platform resources. Currently, academic approaches such as those presented in [2], [3], and [4] support these principles. Industrial frameworks such as Intel CoFluent Studio [5] and Mirabilis Visualsim [6] can also be cited. Our proposal is complementary to such approaches as it attempts to improve efficiency of performance models by combining simulation with formal methods. The approach presented in [7] also considers such a combination to improve simulation efficiency of dataflow oriented architectures. The adopted formal description is based on the real-time calculus (RTC) method initially presented in [8]. Using this formalism, some of the architecture processes are replaced by an equivalent formal model that expresses worst-case bounds on the time sequence of events. This formal model can then be used in conjunction with simulation models to generate events. In our approach, we also consider replacement of parts of the architecture model with an executable model that incorporates formal expressions. The main difference with [7] is that we keep expressions of all internal instants, which allows accuracy to be preserved. Besides, our formal model is obtained directly from the architecture description and not from a prior execution as considered in [7]. In [9], a method is presented to analyze and abstract execution traces of architectures. Performance models are created by eliminating unimportant functions to limit complexity of model behaviour. Our method also abstracts simulated architecture processes but it does not eliminate their influences on architecture performances. In this paper, we point out the influence of the number of abstracted processes on the performance of our method. The result oriented modeling (ROM) approach has been proposed to improve simulation speed-accuracy tradeoff in models of communication resources [10] and operating systems [11]. This approach allows the level of granularity of shared resources models to be increased. It uses optimistic prediction to determine possible architecture evolution instants. In case of disturbing influences such as pre-emption, instants are retroactively adapted during simulation. Retroactive timing correction has also been considered in [12] and [13] to overcome limitations related to the TLM-LT coding style. The method we propose is complementary to these approaches. Architecture characteristics are used to create an intermediate representation. It is then locally executed with no use of the simulator to save simulation events. This method can thus be considered as an extension of the loosely-timed coding style. In this paper, we evaluate possible gains achieved by this approach and we do not address the problem of resource pre-emption. Retroactive timing correction could then be considered to extend this approach. The first author presented some aspects of this modeling approach in [14] but the computation method was not described.

III. Proposed computation method

A. Principles of the approach

The proposed method aims at computing during simulation evolution instants of some parts of a performance model. This method uses expressions of time dependencies among architecture processes. Based on these expressions, an equivalent executable model is used to abstract and replace some of the architecture processes. This equivalent model still presents the same evolution as the abstracted processes from the external viewpoint. However, evolution of the equivalent model is not any more driven by events but depends on locally computed durations. To illustrate these principles we introduce here a didactic example, depicted in Fig. 1. The left part of Fig. 1 represents an architecture model made of five functions and two processing resources. The behaviours of functions F1, F2, F3, and F4 are given using a set of basic communication and computation primitives. The aim of the mapping layer is to correctly manage platform resources when the application model executes, taking into account the concurrency of each platform resource and the defined arbitration and scheduling policies. In the considered example, functions F0, F1, and F2 are allocated to P1, whereas functions F3 and F4 are allocated to P2. Architecture model evolution over the simulation time is illustrated on the lower right part of Fig. 1. Arrows represent some of the events involved during architecture model simulation. Depicted events are related to instants when application functions exchange data through relations. For reasons of clarity we have considered in this example that functions of the application exchange data with a rendezvous communication protocol. Besides, in this example, we have neglected the influence of communication resources of the platform. In Fig. 1, $x_{M2}(k)$ represents the instant at which a simulation event occurs when data is exchanged between F1 and F3 for the $(k+1)$-th time. $T_{i1}(k)$ and $T_{j1}(k)$ represent the execution durations of F1 on P1 for the $(k+1)$-th time. Execution durations are typically variable and can, for example, depend on data size information.

Application of the proposed approach on this example is now considered and illustrated in Fig. 2. Here the equivalent model is the one related to execution of F1, F2, F3, and

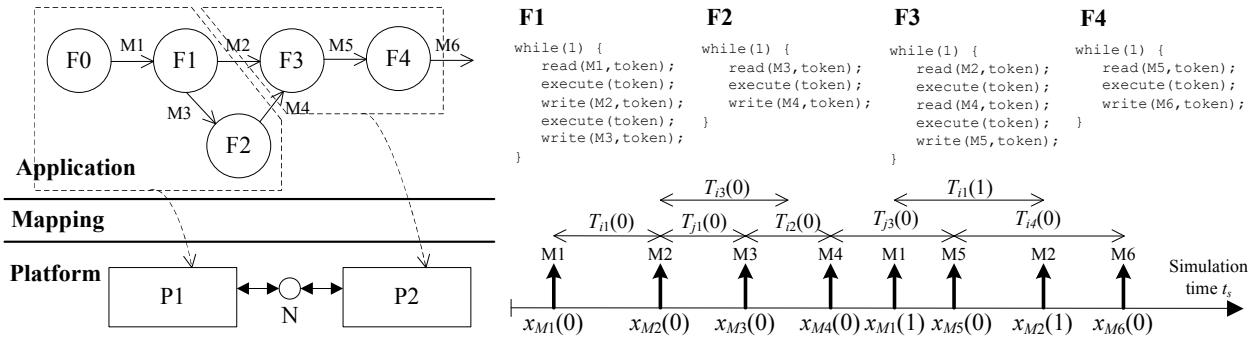

Fig. 1. Illustration of the modeling of an architecture and its event-driven evolution over the simulation time.

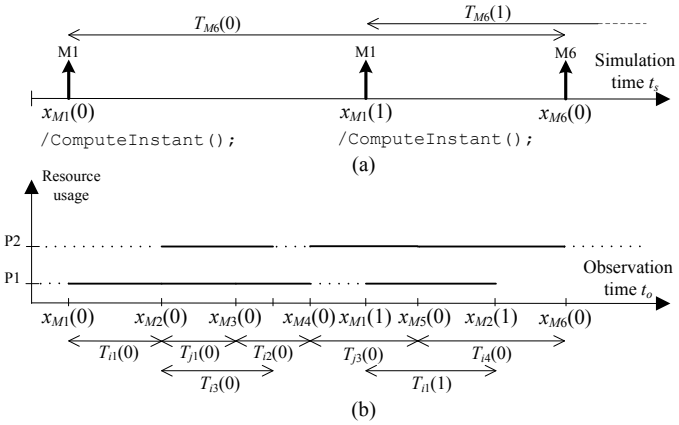

Fig. 2. Simulation of the architecture model with dynamic computation of evolution instants (a) and observation of resource usage over a local time (b).

F4 on processing resources P1 and P2. It still has same input and output relations to be simulated with other parts of the architecture model. The upper part of Fig. 2 illustrates execution of the equivalent model over the simulation time. When a new data is received through relation M1 at instant $x_{M1}(k)$ the value of the output evolution instant $x_{M6}(k)$ is dynamically computed and stored. This computation is performed in zero time, with no advancement of the simulation time. It is denoted by `ComputeInstant()` in Fig. 2. This computation also implies the definition of intermediate instants $x_{M2}(k)$, $x_{M3}(k)$, $x_{M4}(k)$, and $x_{M5}(k)$. Evolution over the simulation time of the equivalent model only depends on the computed value T_{M6} and so intermediate events are saved. The lower part of Fig. 2 represents usage of platform resources P1 and P2. The solid line represents the interval of time during which a processing resource is active. As intermediate instants are computed during model execution, it is still possible to observe usage of resources. This observation is performed using a local time called observation time on Fig. 2. By doing so, evolution of resource usage between $x_{M1}(k)$ and $x_{M6}(k)$ is obtained without using the simulator. Accuracy is thus preserved but with a reduced number of simulation events.

B. Algebraic description of evolution instants

The (max, plus) algebra is used to describe evolution instants of performance models. This theoretical framework was elaborated to favour description and analysis of discrete event systems. Further information about this theory can be found in [15] and [16]. The RTC method used in [7] is also based on this theoretical framework. One of the basic concepts of this theory is that evolution of discrete event systems can essentially be described using two operators:

- addition, denoted by $\otimes$, which expresses a time lag according to a specific duration,
- max, denoted by $\oplus$, which reflects the effect of synchronization among processes.

Based on these two operators a set of equations can be defined to describe time dependencies among evolution instants. To illustrate this idea, we use the example previously introduced in Fig. 1. In the first situation, we assume that P1 can only process one function at a time. This implies that a new data will only be processed through M1 once F2 executed. P2 is considered to be a set of dedicated hardware resources and therefore can compute F3 and F4 at the same time. Evolution instants can thus be described by the following set of equations[1]:

$$x_{M1}(k) = u(k) \oplus x_{M4}(k-1) \quad (1)$$
$$x_{M2}(k) = x_{M1}(k) \otimes T_{i1}(k) \oplus x_{M5}(k-1) \quad (2)$$
$$x_{M3}(k) = x_{M2}(k) \otimes T_{j1}(k) \oplus x_{M4}(k-1) \quad (3)$$
$$x_{M4}(k) = x_{M3}(k) \otimes T_{i2}(k) \oplus x_{M2}(k) \otimes T_{i3}(k) \oplus x_{M5}(k-1), \quad (4)$$
$$x_{M5}(k) = x_{M4}(k) \otimes T_{j3}(k) \oplus x_{M6}(k-1) \quad (5)$$
$$y(k) = x_{M6}(k) = x_{M5}(k) \otimes T_{i4}(k) \quad (6)$$

In (1), $u(k)$ denotes the instant when function F0 tries to produce a data through M1 for the $(k+1)$-th time. In (6), $y(k)$ represents the $(k+1)$-th output evolution instant, that is $x_{M6}(k)$. For example, value $x_{M2}(k)$ is justified by the fact that an event occurs when a data can be produced through M2 only when the computation of F3 is finished or after a duration equal to $T_{i1}(k)$ once a data has been received through M1. Therefore, these expressions are representative of the platform characteristics. As an illustration, if we consider that P2 has also a limited concurrency, $x_{M2}(k)$ should be expressed as follows:

$$x_{M2}(k) = x_{M1}(k) \otimes T_{i1}(k) \oplus x_{M5}(k-1) \oplus x_{M6}(k-1).$$

[1]We recall that in the considered example functions of the application communicate over a rendezvous protocol which implies they wait on each other to exchange data.

Indeed, a new data can be produced through M2 only once F3 and F4 have been processed by P2. In the situation where communications among functions of the application would be performed through FIFO channels additional evolution instants would be necessary to correctly describe the architecture model. Similarly, supplementary equations would be needed to describe the influence of platform communication resources. Considering equations (1) to (6), evolution instants can be formulated with the following matrix expressions:

$$X(k) = A(k,0) \otimes X(k) \oplus A(k,1) \otimes X(k-1) \oplus B(k,0) \otimes U(k) \quad (7)$$

$$Y(k) = C(k,0) \otimes X(k) \quad (8)$$

In (7) and (8), U, X, and Y are the vectors formed respectively by input, intermediate, and output evolution instants. Matrices $A(k,0)$ and $A(k,1)$ reflect dependencies among intermediate evolution instants. $B(k,0)$ reflects dependencies between input evolution instants and intermediate instants. $C(k,0)$ reflects dependencies between intermediate evolution instants and output evolution instants. This illustrates that in some particular cases evolution instants can be represented by a linear expression. General expression of linear evolution of discrete event systems is given by following relations [15]:

$$X(k) = \bigoplus_{i=0}^{a} A(k,i) \otimes X(k-i) \oplus \bigoplus_{j=0}^{b} B(k,j) \otimes U(k-j) \quad (9)$$

$$Y(k) = \bigoplus_{l=0}^{c} C(k,l) \otimes X(k-l) \oplus \bigoplus_{m=0}^{d} D(k,m) \otimes U(k-m) \quad (10)$$

These formulas give an expression of output evolution instants Y according to successive intermediate instants X and input evolution instants U. Of course, not all kind of discrete event systems can receive such a linear expression. For example, presence of conditioning in the evolution of the application implies non linear expression of evolution instants. Besides, other operators than $\oplus$ and $\otimes$ would be needed to capture all the mechanisms related to architecture evolution. For the scope of this paper, we assume that evolution instants can be captured by relations of the following form:

$$X(k) = f(X(k-1), \ldots, X(k-a), U(k), \ldots, U(k-b)) \quad (11)$$

$$Y(k) = g(X(k), \ldots, X(k-c), U(k), \ldots, U(k-d)) \quad (12)$$

f reflects dependencies among intermediate instants and input evolution instants, whereas g reflects dependencies among output evolution instants, intermediate instants, and input evolution instants.

C. Computation method of evolution instants

The aim of action `ComputeInstant()` is to execute set of equations (11) and (12) to determine intermediate and output evolution instants. These equations can be explicitly described and can also be expressed on the basis of an oriented graph. We call such a graph a *temporal dependency graph* as it expresses dependencies among evolution instants. Each node corresponds to a specific evolution instant and weights of arcs define intervals between instants. Traversing this graph leads to successive computation of evolution instants. The temporal dependency graph related to the previous example is represented in Fig. 3. Organization of the temporal dependency graph depicted in Fig. 3 reflects equations (1) to (6). Once input evolution instant $u(k)$ is known, it is possible to successively determine each intermediate instant and output evolution instant $x_{M6}(k)$. Symbol e denotes the identity element for operator $\otimes$. A temporal dependency graph can still be associated to the general equations (9)-(10) and (11)-(12). The number of nodes and arcs depends on the complexity of the modeled architecture. In the situation where conditioning is expressed in the architecture application, additional control statements are required to correctly manage computation. Nevertheless, the computation method is still executed with no use of the simulator. Complexity of such a computation method is related to the number of nodes and arcs that are necessary to determine output evolution instants. In following section, we estimate achieved simulation speed-up by using this computation method. We also evaluate complexity and possible limitations of this method.

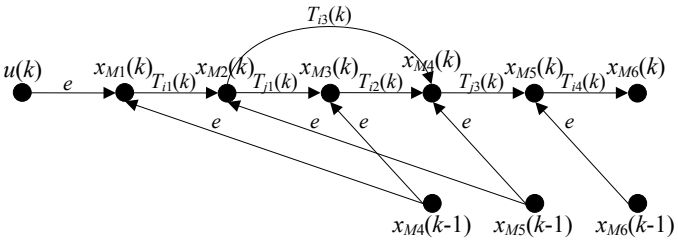

Fig. 3. Temporal dependency graph used to compute evolution instants in the considered example.

IV. Implementation and validation of the computation method

To validate the proposed method we have considered its implementation into a specific modeling framework. Intel CoFluent Studio [5] has been considered because of its graphical interface that allows architecture models to be easily captured. Captured models are then generated in SystemC and execution can be analyzed. The development of a model implementing the proposed computation method can be seen as designing a SystemC module that computes evolution instants from received events, stores output evolution instants, and generates output events accordingly. Fig. 4 gives the structural and behavioural description of the equivalent model related to the considered example. The equivalent model depicted in Fig. 4 is made of two processes. The input process, denoted by Reception, receives successive input events and computes evolution instants through action `ComputeInstant()`. Computed output evolution instants are stored in a local variable, denoted in Fig. 4 by YStored. The output process, denoted by Emission, is activated each time a new output instant has been computed. As depicted in Fig. 4, the $(k+1)$-th output data is produced at instant $x_{M6}(k)$. The implementation of action `ComputeInstant()` corresponds to C++ code developed to correctly process evolution instants using a temporal dependency graph. During model execution this action is performed in zero time, with no advancement of the simulation time. Besides, the organization depicted in Fig. 4 can easily be extended to allow architecture models with multiple inputs and outputs.

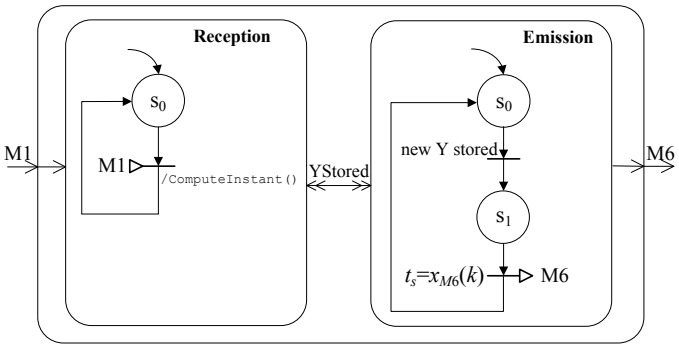

Fig. 4. Structural and behavioural description of the equivalent model in the considered example.

TABLE I. MEASUREMENT OF ACHIEVED SIMULATION SPEED-UP ON DISTINCT ARCHITECTURE MODELS

	Architecture model execution time (s)	Event ratio	Simulation speed-up	Number of nodes
Example 1	22	2.33	2.27	10
Example 2	41.2	4.66	4.47	19
Example 3	59.4	7	6.38	28
Example 4	80.2	9.33	8.35	37

Validation of the approach consists in comparing simulation speed and accuracy among architecture models captured with and without the proposed modeling approach. Accuracy is related to values of models' evolution instants. Execution time of models has been measured on a 2.2GHz Intel Core2 Duo machine. The first experimentation is related to the previously introduced example. Architecture model of Fig. 1 has been developed and compared to its equivalent model to evaluate benefits of the computation method. Both models have been simulated with 20000 data produced through relation M1 with varying data size associated. Evolution instants of both models have been compared and, as expected, remain the same. The first line of Table I gives measurements for this example. In the example of Fig. 1, measured simulation duration is 22 seconds. A simulation speed-up by a factor of 2.27 is achieved by using the proposed method. This value is to be compared with the ratio of simulation events between the two models. This ratio is determined by comparing the number of events that occur when data are exchanged through relations in architecture models. In the considered example this value is equal to 2.33[2]. Other rows in Table I correspond to distinct architecture models with different ratio of events. In all cases simulation accuracy is still preserved but with a significant improvement of the simulation speed. The last column of Table I indicates the number of nodes of each temporal dependency graph used to perform computation of evolution instants. In the considered situations, the computation method presents low influence on the execution time of architecture models.

Complexity of the computation method of evolution instants is directly related to the number of nodes of the temporal dependency graph used to determine evolution instants. A compromise exists between the number of events that can be saved using dynamic computation of evolution instants and the complexity of this method. To evaluate this relation we have considered a varying number of nodes that are required to perform computation of evolution instants. Fig. 5 illustrates the influence of the computation method on the achieved simulation speed-up. In Fig. 5, each curve corresponds to a specific size of vector $X(k)$. This value directly influences the number of events that can be saved using the computation method. The influence of the number of nodes that is needed to perform computation is then observed. In Fig. 5, observation denoted by (*) corresponds to the previously considered example. We observe that the influence of the computation method on the simulation time is negligible for fewer than 100 nodes. Then, achieved simulation speed-up is degraded. For more than 1000 nodes complexity of the computation method leads to a slow down in the simulation. This gives us indication about the limits of the approach.

[2]One could expect a value of 3 for the considered example but implementation into Intel CoFluent Studio implies supplementary events.

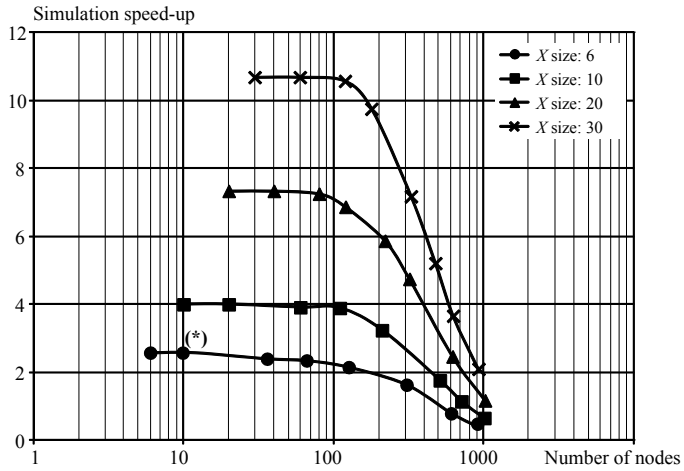

Fig. 5. Evaluation of the influence of the computation method complexity on the achieved simulation speed-up.

V. CASE STUDY

The case study concerns the analysis of required processing resources for a receiver architecture implementing part of the LTE physical layer protocol. LTE protocol is considered for next generation of mobile radio access [17]. Associated baseband architecture demands high computational complexity under real-time constraints and multiprocessor implementation is required. This protocol especially supports high flexibility according to transmitted frames' parameters to adapt to varying user demands. This case study has been detailed in [14] where usage of processing resources for a heterogeneous architecture was studied. In this paper, we focus on application of the proposed computation method to evaluate achieved simulation speed-up. The studied architecture is formed by an application made of eight functions and a platform based on two processing resources. Function behaviour defines computation load implied when the application executes on processing resources. The channel decoding function is considered to be implemented as a dedicated hardware resource whereas other application functions are allocated to a digital signal processor.

We compared two models of the same architecture. The first model is obtained by exhibiting all relations among application functions. The second model is obtained by using the architecture equivalent model incorporating the compu-

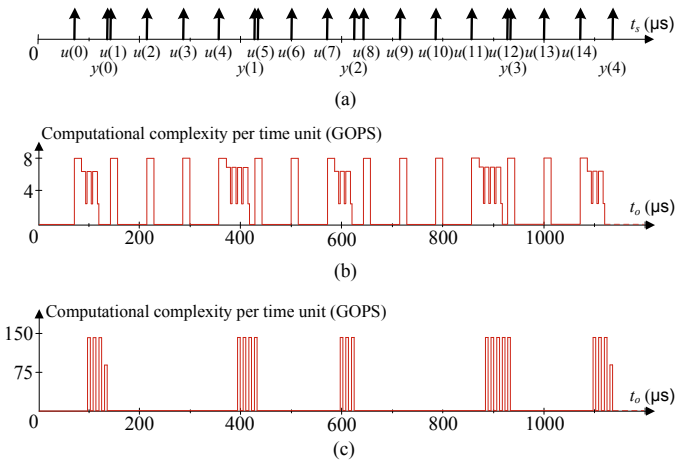

Fig. 6. Observation of studied architecture evolution over the simulation time (a) and over the observation time (b), (c). (b) represents usage of the digital signal processor, (c) represents usage of the dedicated hardware resource.

tation method of evolution instants. In the second model, the temporal dependency graph related to the studied architecture is described to allow output evolution instants to be computed each time a new data symbol is received. This graph contains 11 nodes. Both models are simulated with the same environment that periodically produces data frames with varying parameters. Measurements have been performed under the same conditions as described in the previous section. A simulation speed-up by a factor of 4 has been measured for the simulation of 20000 data symbols, whereas the ratio of events between models is 4.2. Fig. 6 illustrates possible observations. Fig. 6 represents evolution of the architecture model created using the proposed computation method. It illustrates the processing of one complete LTE frame made of 14 symbols and spaced by a period of $71.42\mu s$. The upper part of Fig. 6 illustrates evolution of the model over the simulation time. Once the $(k+1)$-th symbol is received at $u(k)$, the computation method determines intermediate instants and potential output evolution instant $y(k)$. Computation takes into account parameters related to the data frame. The lower part of Fig. 6 represents evolution of platform resources over the observation time. Computational complexity per time unit is observed for each processing resource of the architecture platform. Part (b) of Fig. 6 represents usage of the digital signal processor whereas part (c) represents usage of the dedicated hardware resource. These observations are based on computed intermediate instants and they are performed without using the simulator. The same accuracy is thus obtained as with the initial architecture model but with a significant improvement of the simulation time. Using the computation method described in this paper, we could also consider integration of such a case study in a more complex architecture.

VI. Conclusion

In this paper, we presented a computation method improving the tradeoff between simulation speed and accuracy in performance models of architectures. We have defined the way simulation and formal expression of evolution instants can be combined to allow the number of simulation events to be significantly reduced, leading to improved simulation speed. Influence of the computational complexity of the proposed method has been evaluated and it has been shown that substantial speed-up is achieved in realistic size problems. We are currently developing a tool to support automatic generation of temporal dependency graphs. Also, we plan to extend the computation method to address possible pre-emption of platform resources.

References

[1] IEEE computer society, "IEEE standard SystemC language reference manual," IEEE Std. 1666–2011, Sep. 2011.

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

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

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

[5] Intel. Intel cofluent studio. [Online]. Available: http://www.intel.com/content/www/us/en/cofluent/intel-cofluent-studio.html

[6] Mirabilis. Mirabilis visualsim. [Online]. Available: http://www.mirabilisdesign.com

[7] S. Künzli, F. Poletti, L. Benini, and L. Thiele, "Combining simulation and formal methods for system-level performance analysis," in *Proc. Design, Automation and Test in Europe (DATE'06)*, 2006.

[8] S. Chakraborty, S. Künzli, and L. Thiele, "A general framework for analysing system properties in platform-based embedded system designs," in *Proc. Design, Automation and Test in Europe (DATE'03)*, 2003.

[9] K. Ono, M. Toyota, R. Kawahara, Y. Sakamoto, T. Nakada, and N. Fukuoka, "A modeling method by eliminating execution traces for performance evaluation," in *Proc. of Design, Automation, and Test in Europe (DATE'10)*, 2010.

[10] G. Schirner and R. Dömer, "Result-oriented modeling - a novel technique for fast and accurate tlm," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 26, no. 9, pp. 1688–1699, September 2007.

[11] ——, "Introducing preemptive scheduling in abstract rtos models using result oriented modeling," in *Proc. Design, Automation and Test in Europe (DATE'08)*, 2008.

[12] S. Stattelmann, O. Bringmann, and W. Rosenstiel, "Fast and accurate resource conflict simulation for performance analysis of multi-core systems," in *Proc. Design, Automation and Test in Europe (DATE'11)*, 2011.

[13] K. Lu, D. Muller-Gritschneder, and U. Schlichtmann, "Analytical timing estimation for temporally decoupled tlms considering resource conflicts," in *Proc. Design, Automation and Test in Europe (DATE'13)*, 2013.

[14] A. Barreteau, S. Le Nours, and O. Pasquier, "A state-based modeling approach for efficient performance evaluation of embedded system architectures at transaction level," *Journal of Electrical and Computer Engineering*, vol. 2012, 2012.

[15] F. Baccelli, G. Cohen, G. Olsder, and J. Quadrat, *Synchronization and linearity, an algebra for discrete event systems*. New York: Wiley & Sons Ltd, 1992.

[16] B. Heidergott, G. Jan Olsder, and J. van der Woude, *Max Plus at work. Modeling and analysis of synchronized systems: a course on max-plus algebra and its applications*. Princeton University Press, 2005.

[17] E. Dahlman, S. Parkvall, J. Skold, and P. Beming, *3G evolution: HSPA and LTE for mobile standard*. Amsterdam: Elsevier Academic Press, 2008.