Performance Analysis of Synchronous Models Implementations on Loosely Time-Triggered Architectures

Chung-Wei Lin[†], Marco Di Natale[‡], Haibo Zeng[§], and Alberto Sangiovanni-Vincentelli[†]
EECS Department, University of California at Berkeley[†] Scuola Superiore Sant'Anna[‡] General Motors R&D[§]
E-Mails: cwlin@eecs.berkeley.edu, marco@sssup.it, haibo.zeng@gm.com, alberto@eecs.berkeley.edu

Abstract

Synchronous languages are used in the most popular software and system modeling environments because of the availability of tools for validation and verification by simulation or model checking. A mapping of synchronous models onto Loosely Time-Triggered Architecture (LTTA) that preserves the communication flows has been defined in [5] together with an estimate of lower bounds on the timing performance. The mapping makes use of feedback queues as a backpressure mechanism. In this work, we propose the use of Real-Time Calculus (RTC) as a general model for the timing analysis of such systems, leveraging the work in [3] where RTC was used to analyze multimedia systems with processing pipelines. To apply the analysis to the implementation of synchronous models into LTTA, several additions and fixes to the original analysis are needed. In this paper, we present our early ideas and solutions to these problem.

1. Introduction

Synchronous (reactive) models of computation (SR) are the foundation of several modeling and analysis tools to describe and validate systems functionality, including Simulink and SCADE. SR models are popular because of a strong theoretical backing that allows predictability and formal verification of properties.

However, the preservation of the formal properties of SR models and therefore of the verification results require a formally correct implementation, which is typically obtained using clock synchronization in a time-triggered architecture. A time-triggered framework may be difficult to achieve, especially on distributed wireless networked systems or when functionality needs to be deployed on an existing platform. Therefore, it is important to investigate what are the conditions and the performance for the implementation of a synchronous model into a less constrained architecture, such as the Loosely Time-Triggered Architecture (LTTA) [1]. An LTTA architecture is a distributed architecture in which node clocks are assumed to be *loosely synchronized* (to be compliant with some types of global contract), the communication bus is reliable and delivers messages with a bounded worst-case delay. The architecture nodes provide basic task activation mechanisms and a communication mechanism called *Communication by Sampling* (CbS).

In [5], it is demonstrated how a generic network of SR machines can be implemented by a network of processes communicating through bounded FIFO queues *regardless of the activation times of the FFP processes* if the objective is the preservation of the communication flows. A synchronous model consists of a graph of communicating (possibly infinite-state) Mealy machines. All machines receive streams of data (signals) as input(s) and produce output signals. For simplicity, we assume all the machines at the nodes of the graph perform their computations synchronously and at the same rate, triggered by events coming from a global (logical) clock. The edges in the graph represent the signals communicated from one machine to the other. In order to allow for a well-formed composition, we assume that there is at least one *delay* element in every graph loop. In [5] it is demonstrated how a generic network of SR machines can be implemented by a network of processes communicating through bounded FIFO queues (Figure 1) *regardless of the activation times of the FFP processes*.

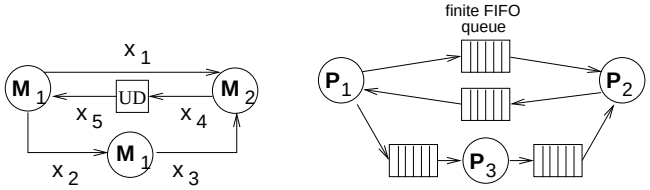

Figure 1. From a synchronous network to an FFP model.

The implementation is correct with respect to the preservation of the communication flows (the signals values), but not with respect to the exact time when they are produced. Each FFP process has the same structure. The computation of the new state and output values is the same as in the original SR machine. Communication occurs through the queues.

A process is triggered by its own clock, but it is executed only if its input buffers are non-empty, and its output buffers are non-full, otherwise it is stalled. In the LTTA network model of [5] activation clocks can be arbitrary or characterized by a lower bound between any two ticks.

The performance analysis is a graph-based approach which involves a reachability graph and the triggering order of processes and uses a *slow triggering policy* which assumes that, in each cycle, the process that is triggered next is the

one that does not enable other firings (the worst-case triggering order). This is why this approach may be too pessimistic. Another problem is that it may suffer exponential numbers of vertices in the analysis graph.

In the domain of real-time schedulability analysis, the Real-Time Calculus (RTC) [2, 4] has been proposed as a general framework for the modelling and analysis of the time properties of embedded SW task systems. RTC has been recently used for the analysis of task systems with feedback [3] (tasks communicating by finite queues). Although the original aim of the analysis are pipelined multimedia systems, we found that after some changes and a few necessary fixes, its concepts are applicable to the analysis of the LTTA implementation of synchronous models.

Our objective in this work is to use RTC to model and analyse the mapping performance of [5]. In order to do so, we need to overcome some limitations of the analysis in [3]:

- We need to derive the effective service curves (more detailed introduction will be in Section 3) for fork and merge topologies, extending the chain topology in [3]. The ability to analyze fork and merge topologies allows to deal with general communication topologies.

- A more accurate and efficient approach to compute the effective service curves of processes is desired. In [3], only the effect of the last process is considered, where the impact of middle processes is ignored, so the resulting effective service curve is too pessimistic.

- Fix a problem in [3] which may compute an effective service curve to be 0, *i.e.*, the process is always idle.

- By applying RTC to the analysis of synchronous models implemented in LTTA, we can improve the analysis presented in [5], which can be quite time-consuming and pessimistic.

The rest of this paper is organized as follows. Section 2 introduces the background knowledge. Section 3 presents our approaches to performing analysis. Finally, conclusion and future directions are in Section 5.

2. Real-Time Calculus and Feedback Control

Min-plus algebra is the basis of RTC. it is based on the definition of a commutative semi-ring, where the operations Min (minimum) and plus (sum) take the place of the addition and multiplication operators we are used to. Given two functions $f(t)$ and $g(t)$ where $t \in \mathcal{R}^+ = [0, \infty)$, the fundamental operators defined by the algebra are:

Definition 1 *The **minimum**, $\oplus$, of $f(t)$ and $g(t)$ is:*

$$(f \oplus g)(t) = \min(f(t), g(t)). \tag{1}$$

Definition 2 *The **convolution**, $\otimes$, of $f(t)$ and $g(t)$ is:*

$$(f \otimes g)(t) = \inf_{0 \le s \le t}(f(s) + g(t-s)). \tag{2}$$

The zero element is the function $f(t) = \infty$ and the unitary element (for the convolution) is the function

$$f^0(t) \quad = \quad \begin{cases} 0, & \text{if } t = 0; \\ \infty, & \text{otherwise;} \end{cases} \tag{3}$$

The dual operation of the convolution is the deconvolution.

Definition 3 *The **deconvolution**, $\oslash$, of $f(t)$ and $g(t)$ is:*

$$(f \oslash g)(t) = \sup_{u \ge 0}(f(t+u) - g(u)). \tag{4}$$

In addition, f^*, the **sub-additive closure** of f is defined.

Definition 4

$$f^* = \bigoplus_{i \ge 0} f^i = f^0 \oplus f^1 \oplus f^2 \oplus \ldots; \tag{5}$$

The Real-Time Calculus (RTC) uses the min-plus algebra to analyze the timing performance of computation and communication nodes. RTC is based on the concepts of input cumulative function, output cumulative function, arrival curves, and service curves. They are formally defined as follows:

Definition 5 *An **input cumulative function** $A(t)$ is the amount of processing time that is requested in the time interval $[0, t]$.*

Definition 6 *An **output cumulative function** $A'(t)$ is the amount of processing time that completes (leaves the system) in the time interval $[0, t]$.*

Definition 7 *The increasing functions $\alpha^u(t)$ and $\alpha^l(t)$ are the **upper and lower arrival curves** of a cumulative function $A(t)$, respectively, if*

$$\alpha^l(t) \le A(t+s) - A(s) \le \alpha^u(t), \forall s, t \in \mathcal{R}^+. \tag{6}$$

The output is produced by the system based on the input request and the availability of processing time, represented by the *service curve*.

Definition 8 *Given a system with input and output flows with cumulative functions $A(t)$ and $A'(t)$, respectively, increasing functions $\beta^u(t)$ and $\beta^l(t)$ are the **upper and lower service curves** of the system, respectively, if*

$$(A \otimes \beta^l)(t) \le A'(t) \le (A \otimes \beta^u)(t), \forall t \in \mathcal{R}^+. \tag{7}$$

The following theorem allows to compute the *output arrival curves*:

Theorem 1 *[2] Given a system with α^u, α^l, β^u, and β^l, the upper and lower output arrival curves are:*

$$\alpha'^u \quad = \quad ((\alpha^u \otimes \beta^u) \oslash \beta^l) \oplus \beta^u; \tag{8}$$

$$\alpha'^l \quad = \quad ((\alpha^l \oslash \beta^u) \otimes \beta^l) \oplus \beta^l, \tag{9}$$

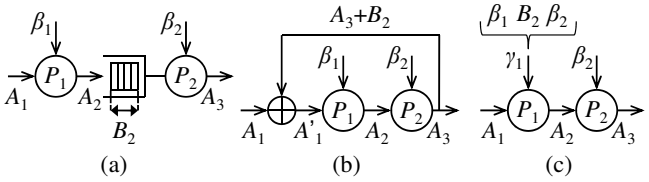

Figure 2. Computing the resulting effective service curve of a sender process connected by a finite FIFO queue to a receiver.

Finally, knowledge of the arrival and service curves for a processing element allows to compute the *maximal backlog*, and the *maximal delay*:

Theorem 2 *[2] Given a system with α^u, α^l, β^u, and β^l, the maximal backlog and the maximal delay are:*

$$b_{\max} = \sup_{t \geq 0}(\alpha^u(t) - \beta^l(t)); \quad (10)$$

$$d_{\max} = \sup_{t \geq 0}\left(\inf_{s \geq 0, \alpha^u(t) \leq \beta^l(t+s)}(s)\right). \quad (11)$$

RTC has been applied to the analysis of systems with feedback [3]. In [3], α^l and β^u are assumed to be 0 and ∞, respectively, so α and β represent the worst-case α^u and β^l, respectively[1]. In order to explain the basic feedback mechanism, two processes P_1 and P_2 connected by a finite queue of length B_2 are shown in Figure 2 (a). To ensure that the backlog of the second process does not exceed B_2, it must be $A_2 - A_3 \leq B_2$ or $A_2 \leq B_2 + A_3$. In order to bound A_2, the first process P_1 must be stalled when the queue is full. This can be obtained by limiting the incoming processed load to the minimum between A_1 and $B_2 + A_3$. This is equivalent to a feedback control before P_1, as shown in Figure 2 (b). The load to P_1 therefore becomes $A'_1 = \min(A_1, B_2 + A_3)$ (the queue is not shown any more since it is modeled by the feedback condition). Given that A_3 is computed based on β_2 and A_2 and the latter is a function of the application of the service curve β_1 to A_1, it seems reasonable to model the need for stalling the processing of A_1 by replacing the service β_1 with a modified or *effective service curve* γ_1 computed as a function of β_1, β_2 and B_2. This dependency is analytically expressed by the theorem:

Theorem 3 *[3] Given a system with two processes, the effective service curve of the first process is:*

$$\gamma_1 = \beta_1 \otimes [\beta_1 \otimes (\beta_2 + B_2)]^*. \quad (12)$$

The authors of [3] proceed to determine an upper bound for the effective service curve of the first process (and the following) in a pipeline configuration as a closed form formula. The effective service curve can then be estimated using the following theorem:

[1]However, the derived effective service curve should be regarded as the upper service curve to ensure that the buffers will never overflow.

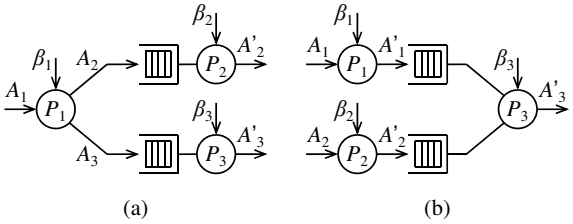

Figure 3. Fork and merge topologies.

Theorem 4 *[3] Given a system of a chain with n processes, the effective service curve of the first process is*[2]*:*

$$\gamma_1 = \beta_1 \otimes \left(\bigoplus_{i=1}^{n} \bigotimes_{j=1}^{i} [\beta_{j-1} \otimes (\beta_j + B_j)]^+\right). \quad (13)$$

We are interested in a more general formulation that allows for the treatment of general graphs as opposed to simple pipelines. Also, when extending the analysis, we found some possible sources of pessimism that could be removed while at the same time simplifying the analysis.

3. Performance Analysis of Process Graphs with Finite Queues and LTTA Implementations

3.1. Fork and Merge Extensions

Handling fork and merge topologies is necessary for the analysis of general graph configurations. The fork topology is shown in Figure 3 (a). When process P_1 finishes its computation, it writes the result to both output buffers. The effective service curve of P_1 computed for the first branch is an upper bound of the service for P_1 that prevents overflow on the corresponding buffer. The same is true for the curve for the second branch. Therefore, the effective service curve of P_1 can be obtained by considering the minimum service resulting from the constraints on both sides. The merge topology is shown in Figure 3 (b). The successor process P_3 can only start its computation when both its input buffers are non-empty. In this case, the problem does not come from the computation of the effective service curve, which is the same as in the chain topology, but from the computation of the output arrival curve of the merging process.

3.2. Effect Propagation

The effective service curve computed in a closed form in [3] is obtained by reasoning by induction. When extending a system with n processes with an $(n+1)$-th queue and process, the effective service curve γ_n is computed (as in Equation (12) by only considering the effect of β_{n+1} on β_n, but leaving $\beta_1, \beta_2, \ldots, \beta_{n-1}$ unchanged, instead of considering their effective values $\gamma_1, \gamma_2, \ldots, \gamma_{n-1}$. Therefore, the formula ignores the effects from $\gamma_2, \ldots, \gamma_{n-1}$ on β_1. The solution in [3] is correct (in the sense that never overflows the buffers) but may be pessimistic, *i.e.*, there may be a solution with higher throughput or lower delay. Preliminary experimental results have shown some examples of this pessimism.

[2]The index i starts from 0 in [3].

Algorithm: Effect-Propagation$(V, E, \beta^u, \beta^l, B)$
1 Sort V by the topology order
2 **for** $i = |V|$ to 1
3 **if** out-degree(v_i)=0
4 $\gamma_i^u = \beta_i^u$
5 $\gamma_i^l = \beta_i^l$
6 **else**
7 $\gamma_i^u = \beta_i^l \otimes [\beta_i^l \otimes (\gamma_{i+1}^l + B_{j+1})]^*$
8 $\gamma_i^l = \beta_i^l \oplus \gamma_i^u$
9 **return** γ^u and γ^l

Figure 4. The Effect-Propagation algorithm.

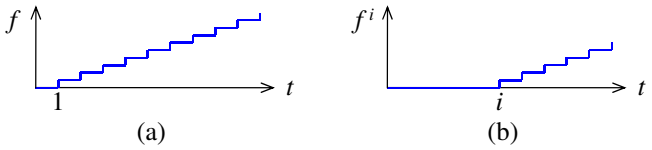

Figure 5. (a) The function f and (b) f^i.

To solve the problem, we yield the compact and elegant closed-form of the solution in [3] for an Effect-Propagation algorithm, as shown in Figure 4. The algorithm follows the reverse topology order and propagates the effects from the last process to the first process (Lines 1–2). In each iteration, it uses the service curve of the next process to update the service curve of the current process (Lines 3–8). The proposed solution has two additional advantages: (1) unlike Equation (13) which can only be applied to a chain topology, the algorithm computes the effective service curves for all processes in a general graph configuration and (2) uses $|V| - 1$ less $\oplus$ operation and the same numbers of other operations, compared with Equation (13).

3.3 Improved Effective Service Curve Analysis

The formula for computing the effective service curve in [3] may compute a function identically equal to 0, *i.e.*, the process is always idle. For example, for the f function shown in Figure 5 (a), the "powers" f^i have the general shape shown in Figure 5 (b), and for an infinite number of products, the result is $f^* = 0$. Hence, whenever, the function $\beta_1 \otimes (\beta_2 + B_2)$ in Equation (12) has an initial interval in which it is zero, the effective service curve computed in this way will be identically 0. In reality, it suffices that $\beta_1(t) = 0$ for $t \in [0, \epsilon]$ for this to happen. We are developing an improved analysis to fix this problem.

4. LTTA Analysis using RTC

RTC can be used for the analysis of LTTA implementations of SR models with different levels of accuracy. At the FFP level, the methods outlined in the previous sections can be used. However, the model needs to be completed by the model of the events activating the FFP processes and the initial service curves. The most common implementation of the FFP processes is as periodic tasks triggered by a stream of periodic events with jitter. The service curve will be linear

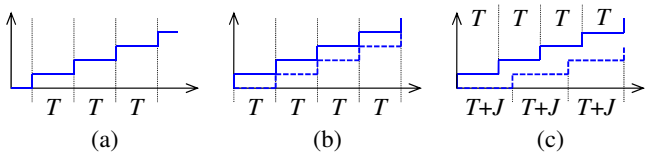

Figure 6. (a) A service curve with clock period T, (b) (upper and lower) service curves with period T and clock drift, and also (c) jitter J.

for the highest priority task ($\beta(t) = t$) and computed according to the priority order for the lower priority tasks. In this case, a periodic service with clock period T can be modelled by a service curve shown in Figure 6 (a). Then, we can modelled the clock drift by the upper and lower services curves shown in Figure 6 (b). They describe the best and worst cases of the service, but they all guarantee that the clock period is T. Furthermore, a jitter J can be modelled by modifying the lower service curve as shown in Figure 6 (c). These curves are lazy so that the effective service curve searching in Section 3.3 is indeed required. A more accurate model of a distributed LTTA implementation should include the delays on the bus communications. In this case, additional processes are needed to represent the network transmission activities and the corresponding delays, on the forward, as well as in the back-pressure communication links.

5. Conclusion and Future Work

Our objective is to use RTC to model and analyse the mapping performance of the mapping of synchronous models into LTTA as in [5]. In order to do so, we need to overcome some limitations of the analysis in [3]. By applying RTC to the analysis of synchronous models implemented in LTTA, we want to improve the analysis presented in [5], which can be quite time-consuming and pessimistic.

References

[1] A. Benveniste, P. Caspi, P. L. Guernic, H. Marchand, J.-P. Talpin, and S. Tripakis, "A protocol for loosely time-triggered architectures," *Proc. of Second Intl Conf. on Embedded Software*, pp. 252–265, 2002.

[2] J.-Y. L. Boudec and P. Thiran, "Network calculus: a theory of deterministic queuing systems for the internet," LNCS 2050, Springer, 2004.

[3] A. Bouillard, L. T. X. Phan, and S. Chakraborty, "Lightweight modeling of complex state dependencies in stream-processing systems," *Proc. of IEEE Real-Time and Embedded Technology and Applications Sym.*, pp. 195–204, 2009.

[4] C.-S Chang, "Performance guarantees in communication networks," Springer, 2000.

[5] S. Tripakis, C. Pinello, A. Benveniste, A. Sangiovanni-Vincentelli, P. Caspi, and M. Di Natale, "Implementing synchronous models on loosely time-triggered architectures," *IEEE Trans. on Computers*, vol. 57(10), pp. 1300–1314, 2008.