Symbolic Performance Analysis of Elastic Systems

Marc Galceran-Oms
Universitat Politècnica de Catalunya
Barcelona, Spain

Jordi Cortadella
Universitat Politècnica de Catalunya
Barcelona, Spain

Mike Kishinevsky
Strategic CAD Lab, Intel Corporation.
Hillsboro, OR USA

***Abstract*—Elastic systems, either synchronous or asynchronous, can be optimized for the average-case performance when they have units with early evaluation or variable latency. The performance evaluation of such systems using analytical methods is a complex problem and may become a bottleneck when an extensive exploration of different architectural configurations must be done.**

This paper proposes an analytical method for performance evaluation using symbolic expressions. Two version of the method are presented: an exact method that has high run time complexity and an efficient approximate method that computes the lower bound of the system throughput.

I. Introduction

Synchronous elastic systems [8], [15], [12] are insensitive to latency variations. The latencies of computations and communications in these designs are decoupled from the functional correctness.

Elastic designs provide a natural framework for design optimizations using variable-latency (or telescopic) units [4] and early evaluation [5], [11], [9]. These methods allow the designer to optimize systems for the typical case rather than the worst case.

Performance analysis of synchronous elastic systems is closely related to the performance analysis of asynchronous systems. Their dynamic behavior can be modeled by marked graphs, a subclass of Petri nets [20]. Different approaches have been taken to analyze the performance of these systems, such as computing the minimum mean-weight cycle [17], performing timing simulations [21], or analyzing systems with min-max functions [14]. However, none of these methods work directly for systems with early evaluation or variable-latency. A method for computing the upper bound of the system throughput has been proposed in [16] using linear programming. An exact method using Markov chains with an exponential complexity is also presented there.

This paper presents an alternative method that combines timing simulations and symbolic algebraic expressions for analyzing the performance of elastic systems with early evaluation and variable-latency units. First, an exact version of this method with an exponential worst case complexity is presented. Then, a fast approximate version of this method is presented that prunes away low impact event correlations and computes a *lower bound* of the system throughput.

Elastic systems can be used to automatically perform architectural exploration [10]. Even for small architectural graphs, there can be hundreds of different configurations whose performance must be analyzed. A fast method to estimate the throughput can significantly speed up such exploration. Furthermore, a lower bound on the throughput provides extra value to the already available upper bound method.

II. Overview

Figure 1(a) shows an example timed marked graph with early evaluation and variable-delay transitions. *Timed multi-Guarded marked Graphs with Variable-delay* (TGVs) will be presented formally in Section III-A. In this Figure, transitions of the marked graph are represented as thick vertical lines, places as the edges between transitions, and tokens as dots over the edges.

When a transition *fires*, it removes one token from each input place and it adds one token to each output place, as shown in Figure 1(b). The enabling of regular transitions is based on *AND-causality*, a transition can fire when all inputs have at least one token. Early evaluation transitions can fire even if some of the inputs have no

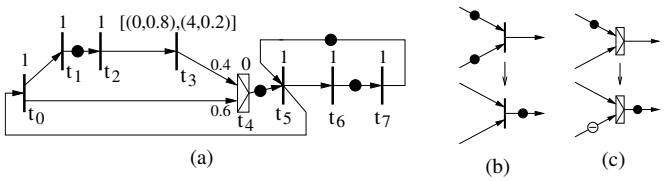


Fig. 1. (a) A TGV with early evaluation and variable-delay, (b) AND-causality firing, (c) early evaluation firing

tokens. Their main purpose is to model multiplexors. The selection of the required branch in the datapath is abstracted in the model by using nondeterministic choices. After each firing, the transition chooses which of the inputs will be required for the next firing. Then, the transition becomes enabled once the selected place has a token available. A *negative token* is inserted into every input place that does not have a token at the moment of transition firing in order to avoid premature firing at the arrival of these irrelevant tokens and to preserve correct behavior. Early evaluation firing is shown in Figure 1(c). Early evaluation transitions are drawn as special boxes, like transition t_4 in Figure 1(a).

Each input of an early evaluation transition is assigned a probability to allow performance analysis. After each firing, early evaluation transitions choose which input will be required for the next firing using these probabilities. In Figure 1(a), the input from t_3 is selected with probability 0.4, and the input from t_0 is selected with probability 0.6.

Each transition is assigned a delay, drawn on top of the transition in Figure 1(a). Once it is enabled, t_1 needs 1 time unit to fire, and transition t_4 fires in zero time. It is possible for a transition to have several possible delays, modeling variable-latency units. Each possible delay is assigned a probability. For example, transition t_3 will need 0 time units to fire with probability 0.8 and 4 time units with probability 0.2. It may represent an ALU with a short operation that finishes combinationally and a long operation that needs 4 clock cycles. The delay of variable-delay transitions is represented as a set of delay-probability pairs: $[(0, 0.8), (4, 0.2)]$.

The throughput of a TGV is defined as the number of times a transition can on average fire per one time unit. Therefore, it is a number between 0 and 1. Performance analysis of choice-free systems has been extensively studied. The throughput can be found by computing the minimum cycle mean [17]. However, to the best of our knowledge, the only work that has studied the performance analysis of systems with early evaluation is [16]. The paper proposes to use linear programming (LP) in order to obtain an upper bound of the throughput. For variable-delay transitions, the average delay of the transition (0.8 for t_3) is used.

We propose to compute the throughput by extracting the probability distribution of the firing times of transitions. Instead of directly deriving the probability distribution for each firing time, the presented method derives a symbolic expression first. Symbolic expressions are able to correctly capture and manipulate correlations between events. Then, the expressions are evaluated in order to obtain the firing times of transitions. Finally, the throughput is computed by extracting the average separation between firing times of the same transition.

The throughput of the system in Figure 1(a) is 0.397, which can be obtained by simulating it. However, the linear programming upper

bound method measures 0.449. Consider t_2 in Figure 1(a). Since it is enabled at the beginning of the simulation and it has delay 1, its first firing occurs at time 1 ($f(t_{2,0}) = 1$, where $f(t_{i,j})$ means the firing time of the j-th firing of transition t_i). Similarly, t_5 and t_7 are initially enabled and they can fire ($f(t_{5,0}) = 1, f(t_{7,0}) = 1$).

Then, t_0, t_3 and t_6 are enabled. Their firing times are obtained by adding their delays to their arrival times: $f(t_{0,0}) = f(t_{5,0}) + 1 = 2$, and $f(t_{6,0}) = f(t_{5,0}) + 1 = 2$. Since t_3 is a variable-delay transition, the firing time becomes probabilistic. Its firing time is 2 with probability 0.8, and 6 with probability 0.2: $f(t_{3,0}) = f(t_{2,0}) + [(0, 0.8), (4, 0.2)] = [(1, 0.8), (5, 0.2)]$

Transition t_4 has an early evaluation enabling function. Therefore, if it chooses the lower edge, which happens with 0.6 probability, its arrival time is the firing time of $t_{0,0}$. If it chooses the upper edge, its arrival time is the firing time of $t_{3,0}$:

$$f(t_{4,0}) = 0.6 \cdot f(t_{0,0}) + 0.4 \cdot f(t_{3,0}) = [(1, 0.32), (2, 0.6), (5, 0.08)]$$

When a transition has more than one input and it is not early evaluated, the maximum of the firing times of the inputs must be computed in order to obtain the arrival time of the transition. For $f(t_{5,1})$, one arrival time is $[(1, 0.32), (2, 0.6), (5, 0.08)]$ from $t_{4,0}$ and the other one is 1 from $t_{7,0}$. If each arrival time has several possible delays, the expression is evaluated by taking each combination, and assigning to the maximum of the delays the multiplication of the probabilities. In this case, $f(t_{4,0}) \geq f(t_{7,0})$.

$$\begin{aligned} f(t_{5,1}) &= 1 + \max(f(t_{4,0}), f(t_{7,0})) \\ &= 1 + f(t_{4,0}) \\ &= [(2, 0.32), (3, 0.6), (6, 0.08)] \end{aligned}$$

We can calculate the average firing time of a transition by computing a mean of the above expression:

$$\overline{f}(t_{5,1}) = 2 \cdot 0.32 + 3 \cdot 0.6 + 6 \cdot 0.08 = 2.92$$

The following table shows the firing times of t_2, the time distance between firings and the average firing time distance , $(f(t_{2,i}) - f(t_{2,0}))/i$.

$\overline{f}(t_{2,0})$	$\overline{f}(t_{2,1})$	$\overline{f}(t_{2,2})$	$\overline{f}(t_{2,3})$	$\overline{f}(t_{2,4})$	$\overline{f}(t_{2,5})$	$\overline{f}(t_{2,6})$	$\overline{f}(t_{2,7})$	$\overline{f}(t_{2,8})$
1	4	5.92	8.43	10.86	13.37	15.88	18.40	20.92
–	3	1.92	2.51	2.43	2.51	2.51	2.52	2.52
–	3	2.46	2.48	2.47	2.48	2.49	2.49	2.50

Only two decimals of the firing time are shown for space reasons. The separation between two consecutive events stabilizes at 2.52, and the average distance between events would also converge to 2.52 if more firings where shown, or if the first firings where ignored to compute this average. By definition, the throughput is the inverse of the average separation between firings of the same transition. Indeed, $1/2.52 = 0.3968$, which is very close to the throughput obtained by simulation, 0.397.

III. Problem Formulation

This section defines the problem to be solved. We assume the reader is familiar with basic Petri nets concepts (see [20] for a tutorial).

A. Multi-guarded Marked Graph

Definition 1 (TGV): A *Timed Multi-Guarded Marked Graph with Variable-Delay* (TGV) is a tuple $N = \langle P, T, Pre, Post, G, \mathbf{m}_0, \delta, \alpha \rangle$ where:

- P is a finite set of places, and T is a finite set of transitions. The *preset* and *postset* of a node $x \in P \cup T$ are denoted as ${}^\bullet x$ and $x^\bullet$. The Petri net is a marked graph: $\forall p \in P, |{}^\bullet p| = |p^\bullet| = 1$.
- $Pre : P \times T \to \mathbb{N} \cup \{0\}$ and $Post : P \times T \to \mathbb{N} \cup \{0\}$ are the *pre* and *post* incidence functions that specify the arc weights. The incidence matrix of the net is $\mathbb{C} = Post - Pre$.
- $\mathbf{m}_0 : P \to \mathbb{Z}$ assigns an initial number of tokens to each place (the initial marking).
- $G : T \to 2^{2^P}$ assigns a set of guards to every transition. The following conditions must be satisfied: a) $\forall g \in G(t)\ g \subseteq {}^\bullet t$; b) $\cup_{g \in G(t)} g = {}^\bullet t$.
- $\alpha : G \to \mathbb{R}^+$ assigns a strictly positive probability to each guard such that for every multi-guarded transition t, $\sum_{g \in G(t)} \alpha(g) = 1$.
- $\delta : T \to \{(d, p)\}$ assigns a list of delay-probability pairs to every transition. It must hold that $d_i \in \mathbb{R}^+ \cup \{0\}$ and for every t, $\sum_{(d,p) \in \delta(t)} p = 1$.

This definition is an extension of the definition of a Timed Multi-Guarded Marked Graph in [16]. The difference is that each transition is assigned a list of possible delays, instead of a single real delay. The delay of each transition t, described by a set of pairs delay-probability $\{(d, p)\}$, represents the number of time units (clock cycles in a latency insensitive design) that t spends computing. If the delay is equal to 0, it means the transition fires immediately, acting like a combinational circuit. The average delay of a set of delay-probability pairs is the mean of the distribution.

$$\overline{\delta}(t) = \sum_{(d,p) \in \delta(t)} d \cdot p \tag{1}$$

The guards of transitions model *early evaluation*. A regular transition has a single guard corresponding to its inputs $G(t) = \{{}^\bullet t\}$. On the other hand, a multiplexor with two possible input places, p_0 and p_1, will have the guards $G(t) = \{\{p_0\}, \{p_1\}\}$. Without losing expressive power, we assume that all early evaluation transitions have one guard for each input. An early evaluation transition with input places $p_0, \ldots p_n$ will have n guards, $\{\{p_0\}, \ldots \{p_n\}\}$. For example, t_4 from Figure 1(a) has two guards, one for the edge (t_0, t_4) and one for the edge (t_3, t_4). Each guard is assigned a probability, $\alpha(g)$, used on simulations and performance analysis to decide how often a guard is selected.

Note that the marking of a place can be negative, modeling the counters that are used in the different early evaluation implementations that have been presented in the literature [9]. Negative tokens appear when an early evaluation transition fires and one of the input places does not have a positive token, as shown in Figure 1(c).

Definition 2 (Firing semantics): The dynamic behavior of a TGV system is determined by its firing rules. The execution of a transition t can be described as follows:

- *Guard selection.* A guard $g(t) \in G(t)$ is selected nondeterministically. Once it is selected, it cannot change until t fires.
- *Delay selection.* A delay $d \in \delta(t)$ is selected nondeterministically. Once it is selected, it cannot change until t fires.
- *Enabling.* Transition t becomes enabled if every place $p \in g(t)$ is positively marked.
- *Firing.* Given a marking $\mathbf{m}$, an enabled transition t can fire leading to a new marking $\mathbf{m'}$ such that $\mathbf{m'} = \mathbf{m} + \mathbb{C}(P, t)$, where $\mathbb{C}(P, t)$ is the column of $\mathbb{C}$ corresponding to t. The firing is performed right after d time units starting from the enabling of the transition.
- *Single-server semantics.* No multiple-instances of the same transition can fire simultaneously. A guard selection is produced for each transition firing, and a transition cannot be enabled again while the previous firing has not completed.

For simplicity, we restrict ourselves to bounded strongly connected graphs.

B. Unfoldings

An *unfolding* of a marked graph [13], [19] is an acyclic marked graph, where each transition corresponds to the firing of one of the transitions in the original marked graph. An unfolding can be divided into *periods*. The i-th period contains the i-th instantiation of each event.

For example, the marked graph in Figure 2(c) is an unfolding of four periods from the marked graph in Figure 2(a) (if dashed arrows are not considered). At the beginning of the unfolding, t_1, t_3, t_0 are enabled.

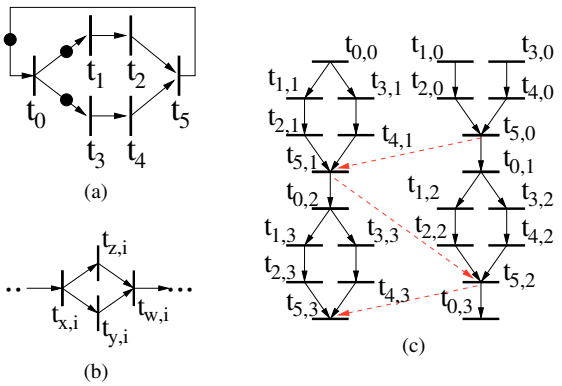


Fig. 2. (a) example TGV, (b) portion of an unfolding with re-convergent paths, (c) 4-period unfolding of Figure 2(a), $t_{k,i}$ stands for the i-th instantiation of transition t_k

Thus, in the unfolding they are the first ones to fire $(t_{1,0}, t_{3,0}, t_{0,0})$, where $t_{k,i}$ means the i-th firing of transition t_k. The firing of t_0 enables the second firing of t_1 and t_3 $(t_{1,1}, t_{3,1})$, and so on. Each transition t_i in the unfolding can be mapped to a transition $\pi(t_i) = t$ in the original marked graph, and the same can be applied to the places. For example, $\pi(t_{1,1}) = t_1$.

C. Single-Server Semantics

In order to ensure *single-server semantics*, some extra edges must be added to the unfolding. These edges make sure that the transition is not re-entrant, i.e. the $(i+1)$-th occurrence of a transition is not enabled before the i-th occurrence has been fired. For each transition t, a single-server semantic place must be added from t_i to t_{i+1}, for all i. These places correspond to the dashed edges in Figure 2(c), which ensure single-server semantics for transition t_5.

Adding single-server semantics places for all transitions can significantly increase the number of edges in the unfolding. Fortunately, it is not necessary to add them for all transitions. It is sufficient to add them for transitions with delay greater than 1 and for transitions which have a fan-in greater than one. It is easy to show that the rest of single-server semantics places become redundant.

D. Throughput

Definition 3 (Steady state throughput): The steady state throughput of a transition t, $\Theta(t)$, of a TGV is defined as:

$$\Theta(t) = \lim_{\tau\to\infty} \frac{\sigma(t,\tau)}{\tau} \tag{2}$$

where τ represents time and $\sigma(t,\tau)$ is the firing count of t at time τ, i.e., it indicates how many times t has fired at time τ.

The firing process is *weakly ergodic* if the limit in equation 2 exists [7]. It can be shown that this limit does exist for a TGV and that it is the same for all transitions [16].

E. Timing Simulation of a TGV

The objective of this work is to determine the *throughput* of an elastic system by determining the *firing time* or *occurrence time* of each transition in an unfolding which is long enough. In [21], the occurrence time of the events of an unfolding is computed in order to obtain the cycle time of an asynchronous circuit with only AND-causality transitions. The firing time of a transition becomes a probability distribution if there is early evaluation or variable-latency in the system.

Definition 4 (Firing time of a transition of an unfolding): The probability distribution for the firing time of a transition t in an unfolded TGV can be defined as follows:

$$\mathbb{P}(f(t) \equiv X) = \mathbb{P}(\delta(t) + \max\{f(t') \mid t' \in {}^\bullet G(t)\} \equiv X) \tag{3}$$

where $\delta(t)$ is a shortcut for $\delta(\pi(t))$.

If t belongs to the set of initial events of the unfolding (${}^\bullet t = \emptyset$), then the probability that the firing time is equal to X it the probability that its delay is X. Otherwise, the probability that $f(t)$ is X is the probability that the chosen delay for t plus the maximum of the firing times of the transitions of the chosen guard is X. The average of the firing time for a transition t in the unfolding is denoted as $\overline{f}(t)$.

Theorem 1 (Steady state throughput using firing times): Let G be a TGV, and t be a transition of G. The steady state throughput of t is:

$$\Theta(t) = \lim_{i\to\infty} \frac{i}{\overline{f}(t_i)} \tag{4}$$

where t_i is the i-th instantiation of t in the unfolding of G.

By definition 3, the throughput is $\sigma(t,\tau)/\tau$, where the time τ tends to infinity. Since t_i is the i-th instantiation of t in the unfolding of G, t will have fired on average i times at time $\overline{f}(t_i)$. Therefore, if $\tau = \overline{f}(t_i)$, then

$$\frac{\sigma(t,\tau)}{\tau} = \frac{i}{\overline{f}(t_i)}$$

□

IV. Max-plus Algebra with Early Evaluation

Instead of directly computing the firing time of each event in the unfolding, our method creates a symbolic expression for each firing time first. The advantage of using symbolic expressions is that they can capture correlations that are lost otherwise. Consider the portion of an unfolding shown in Figure 2(b). Assume that the firing time of $t_{x,i}$ is $f(t_{x,i}) = [(4, 0.2), (5, 0.8)]$. Then, $f(t_{y,i}) = f(t_{x,i}) + \delta(t_{y,i})$ and $f(t_{z,i}) = f(t_{x,i}) + \delta(t_{z,i})$. If both $t_{y,i}$ and $t_{z,i}$ have unit delay, then $f(t_{y,i}) = f(t_{z,i}) = [(5, 0.2), (6, 0.8)]$. In order to compute the arrival time of $t_{w,i}$, one must compute the maximum of $f(t_{y,i})$ and $f(t_{z,i})$. The resulting firing time if each possible combination is taken into account is:

$$\max\left(\begin{bmatrix}(5,0.2)\\(6,0.8)\end{bmatrix},\begin{bmatrix}(5,0.2)\\(6,0.8)\end{bmatrix}\right) = \begin{bmatrix}(\max(5,5),0.04)\\(\max(5,6),0.16)\\(\max(6,5),0.16)\\(\max(6,6),0.64)\end{bmatrix} = \begin{bmatrix}(5,0.04)\\(6,0.96)\end{bmatrix}$$

The previous computation is not correct. Firing times $f(t_{y,i})$ and $f(t_{z,i})$ are correlated, because they both depend on $f(t_{x,i})$. It is never the case that $f(t_{y,i}) \neq f(t_{z,i})$. The correct computation is $\max(f(t_{y,i}), f(t_{z,i})) = [(5, 0.2), (6, 0.8)]$. Using symbolic expressions, some correlations can be structurally detected and fixed. For example, if $\delta(t_{w,i}) = 0$, $f(t_{w,i})$ from Figure 2(b) is:

$$\begin{aligned} f(t_{w,i}) &= 0 + \max(f(t_{x,i}) + \delta(t_{y,i}), f(t_{x,i}) + \delta(t_{z,i})) \\ &= f(t_{x,i}) + \max(\delta(t_{y,i}), \delta(t_{z,i})) \\ &= [(4, 0.2), (5, 0.8)] + \max(1, 1) \\ &= [(5, 0.2), (6, 0.8)] \end{aligned}$$

This section shows how to build and evaluate symbolic expressions that represent the firing time of each transition of an unfolding. Two algorithms are provided. The first one, **A_exact**, computes the exact probability distribution of each firing time, but it has an exponential worst case complexity. The second one, **A_prune**, is an efficient algorithm that obtains a *statistical upper bound* of each firing time. Since A_prune returns *upper bounds* of the actual firing times $\overline{f}(t)$, the throughput of the system obtained from this method will be a *lower bound* of the actual throughput (see equation 4).

A. Definitions

Max-plus algebra [2] has been used in the literature in order to find the cycle time during the steady state of asynchronous circuits with AND-causality. It has also been extended to Min-Max functions in order to perform timing analysis of asynchronous circuits with OR-causality and latch-controlled synchronous circuits with clock schedules [14].

In these papers, AND-causality join structures are translated to max functions and OR-causality to min functions. In order to adapt max-plus

algebra to elastic systems with early evaluation and variable-latency, besides max operation, a new early evaluation operation is needed. Furthermore, leaf variables are sets of delay-probability pairs instead of positive real numbers. While some properties of Max-Min functions are probably lost, some basic useful properties are still maintained in this algebra.

Definition 5 (MPEE expression): A max-plus expression with early evaluation (MPEE), e, is a term in the grammar:

$$e := \delta \mid x \mid e_1 + e_2 \mid e_1 \wedge e_2 \mid e_1^{\alpha_1} \vee e_2^{\alpha_2}$$

where δ is a constant, x is a variable, e_1 and e_2 are MPEE expressions, and α_i are early evaluation probabilities. Constants are sets of delay-probability pairs, and variables must be assigned to sets of delay-probability pairs for evaluation. For every early evaluation operator, the sum of probabilities must be 1.

Operator $+, \wedge, \vee$ represent the sum, max and early evaluation operations for sets of delay-probability. Let us define a grouping operation on sets of pairs by a simple example: $group\{(1, 0.1), (2, 0.3), (1, 0.6)\} = \{(1, 0.7), (2, 0.3)\}$. Now the operations on delay-probability pairs can be defined as follows:

Definition 6 (Addition of two sets of delay-probability pairs): Given two sets of delay-probability pairs, $\delta_1 = \{(d, p)\}$, $\delta_2 = \{(d, p)\}$, their addition is the set of delay-probability pairs $group\{(d_1 + d_2, p_1 \times p_2) \mid (d_1, p_1) \in \delta_1 \wedge (d_2, p_2) \in \delta_2\}$. For example:

$$\begin{bmatrix} (2,0.4) \\ (3,0.6) \end{bmatrix} + \begin{bmatrix} (1,0.8) \\ (2,0.2) \end{bmatrix} = \begin{bmatrix} (3,0.32) \\ (4,0.08) \\ (4,0.48) \\ (5,0.12) \end{bmatrix} = \begin{bmatrix} (3,0.32) \\ (4,0.56) \\ (5,0.12) \end{bmatrix}$$

In all operations, different pairs that have the same delay are grouped into a single pair, and their probabilities are added, like the pairs with delay 4 in the previous example.

Definition 7 (Maximum of two sets of delay-probability pairs): Given two sets of delay-probability pairs, $\delta_1 = \{(d, p)\}$, $\delta_2 = \{(d, p)\}$, their maximum is the set of delay-probability pairs $group\{(\max(d_1, d_2), p_1 \times p_2) \mid (d_1, p_1) \in \delta_1 \wedge (d_2, p_2) \in \delta_2\}$.
For example:

$$\begin{bmatrix} (2,0.4) \\ (3,0.6) \end{bmatrix} \wedge \begin{bmatrix} (1,0.8) \\ (4,0.2) \end{bmatrix} = \begin{bmatrix} (2,0.32) \\ (4,0.08) \\ (3,0.48) \\ (4,0.12) \end{bmatrix} = \begin{bmatrix} (2,0.32) \\ (3,0.48) \\ (4,0.20) \end{bmatrix}$$

Definition 8 (Early evaluation operation): Given two sets of delay-probability pairs, $\delta_1 = \{(d, p)\}$, $\delta_2 = \{(d, p)\}$, and two real numbers, α_1, α_2 such that $0 \le \alpha_1, \alpha_2 \le 1$ and $\alpha_1 + \alpha_2 = 1$, the early evaluation operation results in the set of delay-probability pairs $group\{\bigcup_{i\in\{1,2\}}\{(d, p \times \alpha_i) \mid (d, p) \in \delta_i\}\}$. For example:

$$\begin{bmatrix} (2,0.4) \\ (3,0.6) \end{bmatrix}^{0.1} \vee \begin{bmatrix} (1,0.8) \\ (4,0.2) \end{bmatrix}^{0.9} = \begin{bmatrix} (2,0.04) \\ (3,0.06) \end{bmatrix} \bigcup \begin{bmatrix} (1,0.72) \\ (4,0.18) \end{bmatrix} = \begin{bmatrix} (1,0.72) \\ (2,0.04) \\ (3,0.06) \\ (4,0.18) \end{bmatrix}$$

All operations have arity two, but they can be easily extended to arity n. We assume that $+$ has a higher binding than $\wedge$ or $\vee$. Operations $\wedge$ and $\vee$ are associative and commutative. Addition distributes over both $\wedge$ and $\vee$.

$$\begin{aligned} e_1 + (e_2 \wedge e_3) &= e_1 + e_2 \wedge e_1 + e_3 \\ e_1 + (e_2^{\alpha_1} \vee e_3^{\alpha_2}) &= (e_1 + e_2)^{\alpha_1} \wedge (e_1 + e_3)^{\alpha_2} \end{aligned} \tag{5}$$

B. Building MPEE Expressions

Given an unfolding of a TGV, an MPEE expression is built for each transition. Expressions have a unique pointer in memory, using a technique similar to BDD databases [6]. This way, memory usage is lower and evaluation of expressions is faster, since results can be reused. This is useful, for example, when several transitions share the same set of inputs, since the expression built to compute the arrival time will be shared. Notice that it will cause the expression to form an acyclic graph instead of a tree.

Algorithm 1: BuildMPEE

Input: A TGV G and a transition t
Output: An MPEE expression for the firing time of t

```
if t ∈ BuiltExpressions then
    return BuiltExpressions[t]
arrival = ∅
for p ∈ •t do
    e = BuildMPEE[•p]
    arrival = arrival ∪ {e}
if Early(t) then
    e_a = MakeEarly(arrival, Probs(t))
else
    arrival = SimplifyMax(arrival)
    e_a = MakeMax(arrival)
e_a = FactorOut(e_a)
e = MakePlus(e_a, MakeLeaf(δ(t)))
BuiltExpressions[t] = e
return e
```

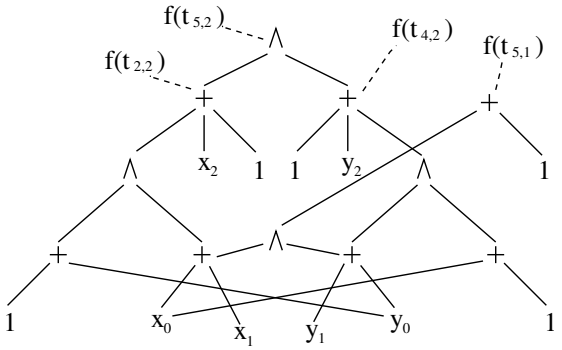


Fig. 3. Graph representation of an MPEE expression for unfolding in Figure 2(c). x_k represents $\delta(t_{2,k})$ and y_k represents $\delta(t_{4,k})$

Algorithm 1 calls the following procedures:

- *Early*. Checks whether the given transition is early evaluation.
- *Probs*. Returns the guard probabilities for the input transitions.
- *MakeLeaf*. Returns an expression associated with the delay of a transition. If it is not a variable-delay transition, then a constant is returned. Otherwise, a new variable is generated.
- *MakePlus, MakeMax, MakeEarly*. Given a list of expressions, a new expression is created which is either the addition, max or early of the given list. MakeEarly receives a list of probabilities. If an expression with the same operators and operation already exists in memory, then the existing expression is returned. A lower and upper bound for the expression is computed and recorded. Some basic simplifications are done, e.g., constants are grouped ($0 \wedge 1 \wedge e_0 = 1 \wedge e_0$).
- *SimplifyMax*. Given a list of expressions, it checks whether some of them is subsumed by the rest when computing the maximum of the expressions. It uses the upper and lower bounds of the expressions.
- *FactorOut*. Given an early or max expression, it applies the distributive property in order to factor out common sub-expressions.

Given the following delays $\delta(t_0) = \delta(t_5) = 0, \delta(t_1) = \delta(t_3) = 1$, $\delta(t_2) = \delta(t_4) = [(1, 0.5), (2, 0.5)]$; the firing time of $t_{5,2}$ ($f(t_{5,2})$) in Figure 2(c) is described by the expression graph in Figure 3. In this Figure, x_k is a short-cut for $\delta(t_{2,k})$ and y_k is a short-cut for $\delta(t_{4,k})$. It can be seen how expressions are reused to compute different firing times, such as $f(t_{5,2})$ or $f(t_{5,1})$. In fact, there is a unique expression graph which receives pointers from the firing times of all transitions. They are not drawn in this Figure in benefit of simplicity.

C. Lower and Upper Bounds of Expressions

In order to discard redundant expressions when a new max expression is added, the lower and upper bound of each expression is recorded. These bounds can be computed using the following rules:

$$
\begin{array}{lll}
e = \delta(t) & \rightarrow & ub(e) = ub(\delta(t)) \quad lb(e) = lb(\delta(t)) \\
e = e_1 + e_2 & \rightarrow & ub(e) = ub(e_1) + ub(e_2) \quad lb(e) = lb(e_1) + lb(e_2) \\
e = e_1 \wedge e_2 & \rightarrow & ub(e) = \max(ub(e_1), ub(e_2)) \quad lb(e) = \max(lb(e_1), lb(e_2)) \\
e = e_1 \vee e_2 & \rightarrow & ub(e) = \max(ub(e_1), ub(e_2)) \quad lb(e) = \min(lb(e_1), lb(e_2))
\end{array}
$$

For leaf expressions, the upper and lower bounds are the upper and lower bounds of the corresponding set of delay-probability pairs. If the expression is a sum of the expressions, then the upper bound is the sum of upper bounds, and the lower bound the sum of lower bounds. The same rationale can be applied to the upper and lower bound of a max operation. For an early evaluation operation, the upper bound corresponds to the case where the selected branch is the one with the greatest upper bound. Similarly, the lower bound corresponds to the case where the selected branch is the one with the lowest lower bound.

The SimplifyMax operation discards redundant expressions given a list of expressions whose maximum is going to be computed. In order to do so, the maximum of the lower bounds is computed (max_lb). Then, for each expression with bounds lb, ub, if $ub < max\_lb$, the expression is discarded.

Although more sophisticated methods to discard redundant branches have been tried, none of them was fast enough so that it could be used while the MPEE expressions are generated. For example, an SMT [3] solver was called in order to discard redundant expressions. This solution could take into account correlations between expressions in order to determine if an expression is redundant (e.g., it could detect that $a+c$ and $b+c$ are redundant in $a+b+c \wedge a+c \wedge b+c$), but it was computationally too expensive, specially when the depth of the MPEE expression is very big.

D. Exact Evaluation of MPEE Expressions

Once an MPEE expression has been built for every transition, the expressions must be evaluated in order to be able to obtain the throughput. The accuracy of the solution decreases for every max operation in which the operands are correlated. In this case, the result does not correspond any more to the result that would result by correctly evaluating equation 3.

The cause of these correlated expressions are re-convergent paths, like the one shown in Figure 2(b). Both [1] and [18] identify this problem when doing statistical timing analysis of circuits and propose exact solutions with exponential time cost. They identify nodes that are the sources of re-convergent paths, and then they obtain an exact result by evaluating each possible delay by itself in the sources of re-convergent paths. Given an MPEE expression, it is possible to identify which sub-expressions are the sources of re-convergent paths by traversing the expression.

Definition 9 (Correlated Expression): Given an MPEE expression $e = e_1 \wedge e_2 \wedge \ldots e_n$, an expression e' is correlated with regard to e if there exists at least two sub-expressions of e (e_i, e_j), such that there is a path in the expression graph from e_i to e' and from e_j to e'.

Correlated expressions are the sources of a re-convergent path in the expression graph, which means there is a re-convergent path in the unfolding which could not be factored out. We are only interested in the correlated expressions that may introduce a set of delay-probability pairs with more than one element, even if their inputs are a single delay-probability pair. These are leaf expressions which represent variable-delay transitions, and early evaluation expressions. The set of correlated expressions for a given max expression can be found by traversing its expression graph.

Symbolic MPEE expressions may alleviate the problem of re-convergent paths since expression manipulation can factor out common expressions in a max operation. However, there are some cases where this is not possible. For example, in Figure 3, either x_2 and y_2 are correlated expressions or x_0 and y_0 are correlated expressions. It is not possible to transform the expression using the algebra properties so that none of them is a correlated expression.

Algorithm 2: Evaluate

Input: Expression e, an integer i, a list of CorrelatedExpressions and a probability *prob*

```
if i < 0 then
    Eval(e)
    Scale(e,prob)
else
    ce = CorrelatedExpressions[i]
    if Early(ce) then
        for each guard (g, p) of ce do
            SelectGuard(ce, g)
            Evaluate(e, i − 1, CorrelatedExpressions, prob × p)
    else
        ce is a leaf expression with variable-delay
        for each delay (d, p) of ce do
            SelectDelay(ce, d)
            Evaluate(e, i − 1, CorrelatedExpressions, prob × p)
```

To obtain the exact result of an expression e even if there are correlated expressions, we must do a case by case evaluation. For each possible selection of delays in the correlated expressions, e must be evaluated and its result scaled to the selection probability. Algorithm 2 shows how evaluation case by case can be computed using a recursive algorithm. The top most call must be Evaluate(e, *size*(CorrelatedExpressions), CorrelatedExpressions, 1), where CorrelatedExpressions is the list of correlated expressions with regard to e.

For each correlated expression, it selects all possibilities one by one and calls the algorithm recursively. The accumulated probability is kept. Once all the correlated expressions have been traversed, the procedure *Eval* is called, which evaluates the expression using definitions 6, 7 and 8. *Eval* is a recursive function that will evaluate sub-expressions if it is needed. After evaluation, the partial result is stored after scaling it with the accumulated probability *prob*.

For the example in Figure 3, the correlated expressions are x_0 and y_0. If all variables x_i and y_i have a delay equal to $[(1, 0.5), (2, 0.5)]$, the four different evaluations to do are:

Delays	Result
$x_0 = 1, y_0 = 1$	$[(4, 0.0625), (5, 0.5), (6, 0.4375)]$
$x_0 = 1, y_0 = 2$	$[(5, 0.125), (6, 0.625), (7, 0.25)]$
$x_0 = 2, y_0 = 1$	$[(5, 0.125), (6, 0.625), (7, 0.25)]$
$x_0 = 2, y_0 = 2$	$[(5, 0.0625), (6, 0.5), (7, 0.4375)]$

Each combination has a probability of 0.25. The final probability distribution is obtained by scaling each distribution by its probability and combining them together: $[(4, 0.015625), (5, 0.203125), (6, 0.546875), (7, 0.234375)]$. The final average firing time is 6.0.

After evaluation for the case $x_0 = 1, y_0 = 1$, the evaluation for the case $x_0 = 1, y_0 = 2$ is performed. Notice that not all expressions must be reevaluated, only the ones that depend on the value of y_0, since x_0 has not changed. For this example, this would save 5 evaluations. This optimization is taken into account by the evaluation algorithm.

Algorithm A_exact(k) creates an unfolding of k periods. Then, Algorithm 1 is called in order to build the expression graph for all the

transitions in the unfolding. Finally, Algorithm 2 obtains the probability distribution for the firing time of each transition. Assuming k is large enough, the throughput can be obtained by using equation 4. The running time of this method is exponential with regard to the number of correlated expressions in the expression graph.

E. Upper Bound Evaluation of MPEE Expressions

For most examples, A_exact is infeasible because correlations cannot be factored out when building the expressions and the runtime becomes exponential. Algorithm A_prune introduces a couple of heuristics that significantly reduce its complexity. First, the size of the sets of delay-probability pairs must be kept small enough so that operations do not take too much time. Second, some correlated variables and early evaluation expressions can be ignored and computed as if they did not add re-convergent paths. A_prune computes an upper bound of the firing times of each transition. Therefore, the obtained throughput will be a lower bound of the actual throughput.

1) Size of Sets of Delay-Probability Pairs: As the expressions grow and have a larger depth, the number of possible delays in the solution also grows. Computing max and plus operations has a quadratic cost on the size of the operands. Thus, if $\delta_0 \wedge \delta_1$ is computed, and δ_0 is a set of delay-probability pairs with 20 possible delays, and δ_1 has another 20 possible delays, 400 max operations plus 400 multiplications of probabilities will be computed.

Corner cases tend to have smaller and smaller probabilities. If a variable-delay transition t has delay 1 with probability of 0.1, after 10 periods of the unfolding, there will be one delay-probability pair with probability 10^{-10} corresponding to the case where all instantiations of t selected delay 1.

The probability distribution of the firing time of transitions after some periods of the unfolding typically has the form of a mountain chain, where the greatest delays and the smallest delays have the smallest probabilities. One optimization that can reduce the running time with no significant cost on the accuracy of the results is to remove delays with low probability. Neighboring expressions with probabilities lower than a threshold are grouped together into a single pair with a value that does not change the average of the set.

For example, consider the delay $[(10, 10^{-4}), (11, 1.5 \times 10^{-4}), (12, 0.1), (13, 0.2), \ldots]$. If the threshold is 10^{-3}, then the first two delays should be grouped into the third one. The resulting delay d and probability p are computed as follows:

$$p = 10^{-4} + 1.5 \times 10^{-4} + 0.1 = 0.10025$$
$$d = (10 \times 10^{-4} + 11 \times 1.5 \times 10^{-4} + 12 \times 0.1)/p = 11.9965$$

A high threshold may force too much grouping, which may actually slow down the computation. Empirically, this value has been set to 10^{-20}. To ensure that results do not grow too much, if the size of the set is larger than some value, for example, 64, some extra grouping may be performed with a larger threshold.

2) Ignoring Correlated Expressions: Some correlations may be completely ignored. The condition to end the recursion in Algorithm 2 ($i < 0$) can be changed to:

$$i < max(0, size(\text{CorrelatedExpressions}) - max\_correlated\_expr + 1)$$

If $max\_correlated\_expr = 0$, then all correlations are ignored by Algorithm 2, which directly evaluates the expression. If $max\_correlated\_expr = 2$, then the only the two first correlated expressions in CorrelatedExpressions will be considered. Notice that even if $max\_correlated\_expr$ is 0, some correlations are still taken into account, since they are captured and simplified away when building the MPEE expression in Algorithm 1.

For the example in Figure 3, if only correlations due to x_0 are taken into account, only two evaluations must be done:

Delays	Result
$x_0 = 1$	$[(4, 0.015625), (5, 0.296875), (6, 0.5625), (7, 0.125)]$
$x_0 = 2$	$[(5, 0.09375), (6, 0.5625), (7, 0.34375)]$

Since each combination has a probability of 0.5, the final result is $[(4, 0.0078125), (5, 0.1953125), (6, 0.5625), (7, 0.234375)]$, which has an average of 6.02. The exact average firing time was 6. If no correlations are taken into account, then the final result $[(4, 0.003906), (5, 0.1875), (6, 0.574219), (7, 0.234375)]$, which has an average of 6.04. In general, the average firing time increases as the number of correlations considered decreases.

3) Upper Bound Theorem: As the previous example shows, ignoring correlations yields an upper bound of the firing time. Since the throughput of an elastic system is computed using $i/f(t_i)$, the throughput computed ignoring correlated expressions will be a *lower bound* of the real throughput.

Definition 10 (Definition 5 from [1]): A cumulative distribution function (CDF) $Q(x)$ is a statistical upper bound of another CDF $S(x)$ if and only if for all x, $Q(x) \leq S(x)$.

Using definition 10, we can state that, for each transition t_i of the unfolding, the firing time probability distribution that A_prune computes for t_i is a statistical upper bound of the actual firing time probability distribution of t_i.

Theorem 2: Let e be an MPEE expression representing the firing time of transition t_i of the unfolding. Let $Q(x)$ be the CDF of the probability distribution computed by using definitions 6, 7 and 8 on e, and let $S(x)$ be the probability distribution which is the solution of equation 3 for transition t_i. $Q(x)$ is a statistical upper bound of $S(x)$.

The proof for this theorem is omitted due to space limitation. It is done using a technique similar to the proof of Theorem 2 in [1] with an extension to handle the early evaluation operation.

V. Number of Periods of the Unfolding

The final question that needs to be answered is how many times the TGV should be unfolded, for both A_exact and A_prune algorithms. In [21], an upper bound of the periods of an unfolding is found, so that it can be guaranteed that the exact cycle time is correctly computed. Unfortunately, variable latency and early evaluation make this result invalid, although some other results from the paper still hold. A dynamic algorithm is used in order to determine the number of periods for the unfolding. This section presents some theoretical background to justify this dynamic algorithm.

A. Convergence of A_exact

Definition 11 (Average separation between occurrences): The average separation between two occurrences of the same transition t, t_i and t_j, is defined as:

$$\Delta_i(t_j) = \frac{\overline{f}(t_j) - \overline{f}(t_i)}{j - i} \tag{6}$$

The first firing times of a timing simulation can have some perturbations because the steady state has not been reached yet. However, using a technique similar to [21], it can be proven that for any transition t, after some initial number of periods i_0,

$$\max_{\forall j > i \geq i_0} \{\Delta_i(t_j)\} \leq 1/\Theta \tag{7}$$

That is, the throughput estimate computed as an inverse of the average separation between occurrences over some length of simulation is an upper bound on the exact throughput.

The limit from the throughput definition (equation 4) exists, as pointed out in Section III-D. Equation 7 is monotonically improving the bound because of the maximum operation. The longer the unfolding, the closer equation 7 gets to the exact throughput. While this equation may only reach its limit in the infinity, the following property can

be shown based on a desired error $\epsilon > 0$ for computing the average separation between the occurrences:

$$\forall \epsilon > 0,\ \exists\, k > 0,\ \max_{\forall k \geq j > i \geq i_0} \{\Delta_i(t_j)\} \geq (1-\epsilon) \cdot 1/\Theta \tag{8}$$

This equation justifies the following algorithm. Iteratively, a number of periods is added to the unfolding and then the firing times for the new transitions are computed. Let A_exact(k) be $\max_{\forall k \geq j > i \geq i_0} \{\Delta_i(t_j)\}$, the result obtained by running A_exact algorithm on a k-period unfolding. First, A_exact($T/2$) is computed, where T is the number of tokens in the TGV. Next, $T/2$ periods are added at each iteration, until the result stabilizes, i.e., A_exact($i \cdot T/2$) + $(1-\epsilon) \geq$ A_exact($(i+1) \cdot T/2$), where ϵ is a small number that can be tuned to decide the acceptable error of the algorithm. The number of tokens T is chosen since it scales together with the size and the complexity of the TGV.

B. Convergence of the Upper Bound Method, A_prune

The dynamic method to decide when to stop adding periods to the unfolding can also be applied to A_prune algorithm. For every transition in the unfolding t_i, A_prune computes a firing time probability distribution $f^{ub}(t_i)$ which is a statistical upper bound of $f(t_i)$. There are two important facts to consider::

1) By construction, the upper bound of both distributions is the same, $ub(f^{ub}(t_i)) = ub(f(t_i))$.
2) Since each max expression which has correlated expressions adds some extra overhead to the upper bound, and no other type of operation modifies the accuracy of the result, $\overline{f^{ub}}(t_i) = \overline{f}(t_i) + k \implies \overline{f^{ub}}(t_{i+1}) \geq \overline{f}(t_{i+1}) + k$

Due to the second item, the estimation of the average occurrence distance based on the upper bound, $\overline{f^{ub}}(t_i)$, cannot decrease with iterations, and due to the first item, it is always limited from above. Hence the difference between the actual firing time and its computed upper bound is bounded, and it cannot decrease if more periods are added to the unfolding.

Let Θ^{lb} be the throughput returned by applying A_prune to an infinite unfolding. Considering that $\overline{f^{ub}}(t_i) \geq \overline{f}(t_i)$, it can be derived that $\Theta^{lb} \leq \Theta$. If the previously defined dynamic method is applied to A_prune, it will return a throughput Θ^{lb}/ϵ, where $\epsilon \leq 1$. It must be ensured that $\Theta^{lb}/\epsilon \leq \Theta$, because our objective is to obtain a lower bound of the throughput. Since A_prune is a fast algorithm, the accuracy parameter ϵ in the dynamic algorithm can be set to a small enough value so that the resulting throughput will either be a lower bound of the Θ or it will be equal to Θ for enough significant digits.

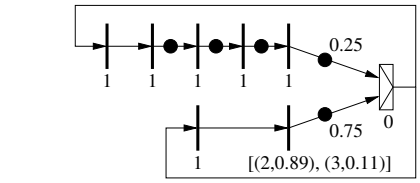
C. Example

Figure 4(b) shows the throughput of the TGV in Figure 4(a) with different number of unfoldings and different correlation strategies. In Figure 4(a), the number below each transition is its delay. There is one early evaluation transition and one variable-delay transition. The throughput obtained by simulating the elastic system is 0.537. The throughput obtained by the linear programming method in [16] is 0.618.

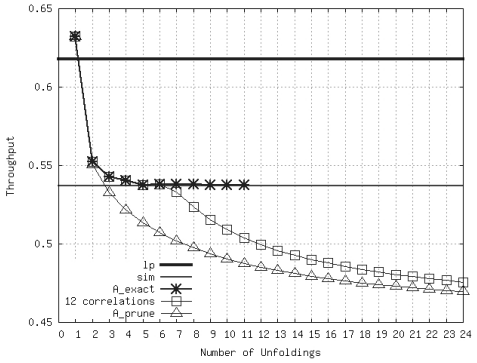
The two horizontal lines in Figure 4(b) correspond to the simulation throughput and the LP throughput. The other three lines correspond to A_exact(k), A_prune(k), and a third algorithm where *max_correlated_expr* (presented in Section IV-E2) has been set to 12. The reader should keep in mind that the Figure shows the throughputs, which are the inverse of the average firing times.

It can be seen that A_exact(k) complies with equation 7. The computed throughput is always greater than the simulation throughput, and after 5 periods it converges, returning the exact throughput.

A_prune is initially over the throughput, but after 3 periods it crosses the line representing the simulation throughput. As it has been explained, it is possible that this happens if the unfolding does not have



(a)



(b)

Fig. 4. (a) TGV with early evaluation and variable-delay (b) Throughput of Figure 4(a) computed using different numbers of unfoldings and different correlation strategies

enough periods. Afterwards, it converges at 0.470, which is indeed a lower bound of the throughput.

While the firing time expression has less than 12 correlated expressions, which happens for the first 6 periods, the line for *max_correlated_expr* = 12 behaves like A_exact. Afterwards, it starts losing accuracy, and it converges at 0.475.

VI. Experimental Results

This section presents a set of experiments with random graphs. The relative error of the different methods is evaluated.

A. Random Graphs

To show the accuracy of the presented method, a set of experiments has been performed on sequential circuits from the MCNC benchmarks. For each circuit, the graph structure was extracted, and the largest strongly connected component was saved. Then, each edge was assigned a token with a random probability between 0.25 and 0.6. Next, each transition was assigned unit delay with a random probability greater than 0.5, otherwise its delay was set to zero. Variable delays were added to 10% of the nodes. These nodes were assigned two random possible delays between 0 and 10. Each delay was assigned a random probability. Finally, 25% of the nodes with 2 or more inputs were configured as early evaluated. The probability of each input was generated randomly.

Table I shows the list of graphs and their throughputs. For each graph, three versions were saved. One with only variable-delay nodes, one with only early evaluation nodes, and one with early evaluation and variable-delay. We compare our results against the upper bound method (LP) in [16] that uses linear programming.

The maximum number of correlations taken into account by the evaluation algorithm was set to 0 in order to obtain the fastest possible run-times. When building the expressions, some correlations can already be factored out. For the case where there is only early evaluation and no variable-delay, the average errors for the upper bound method and the lower bound method are 6.8% and 3.5% respectively. Both methods are quite accurate, although there are two graphs where the error of the upper bound method is over 10%.

The LP method is less accurate on graphs with variable-delay nodes since it does not know the possible delays of these transitions. For

TABLE I
Experimental results. N: nodes, E: edges, EV: early evaluation nodes, VD: variable-delay nodes, T: tokens, D: nodes with $\delta > 0$, UB: upper bound throughput (LP), SIM: simulation throughput, LB: lower bound throughput, Δ_{UB}: UB error w.r.t. SIM, Δ_{LB}: LB error w.r.t. SIM

Graph	N	E	EV	VD	T	D	UB	SIM	LB	Δ_{UB}	Δ_{LB}
	Only Early Evaluation										
s420	9	10	1	0	2	4	0.500	0.500	0.500	0%	0%
s838	9	10	1	0	2	4	0.412	0.412	0.412	0%	0%
s208	9	10	1	0	5	8	0.765	0.732	0.656	5%	10%
s27	18	28	4	0	10	28	0.619	0.472	0.429	31%	9%
s382	48	66	6	0	15	39	0.254	0.253	0.253	0%	0%
s526	56	77	6	0	21	46	0.250	0.239	0.235	5%	2%
s400	54	74	8	0	38	60	0.713	0.702	0.696	2%	1%
s444	70	94	12	0	33	45	0.867	0.805	0.801	8%	0%
s386	59	142	11	0	53	129	0.250	0.217	0.199	15%	8%
s1488	180	619	47	0	139	340	0.333	0.321	0.305	4%	5%
	Only Variable Delay										
s420	9	10	0	1	2	4	0.287	0.287	0.264	0%	8%
s838	9	10	0	2	2	4	0.107	0.107	0.101	0%	6%
s208	9	10	0	1	5	8	0.322	0.322	0.308	0%	4%
s27	18	28	0	2	10	28	0.333	0.199	0.172	67%	14%
s382	48	66	0	5	15	39	0.161	0.133	0.126	21%	5%
s400	54	74	0	7	38	60	0.476	0.265	0.251	80%	5%
s526	56	77	0	3	21	46	0.182	0.175	0.172	4%	2%
s444	70	94	0	5	33	45	0.526	0.272	0.229	93%	16%
s386	59	142	0	3	53	129	0.143	0.143	0.143	0%	0%
s1488	180	619	0	7	139	340	0.152	0.148	0.127	3%	14%
	Both Early Evaluation and Variable Delay										
s420	9	10	1	1	2	4	0.372	0.329	0.288	13%	12%
s838	9	10	1	2	2	4	0.207	0.107	0.101	93%	6%
s208	9	10	1	1	5	8	0.618	0.538	0.470	15%	13%
s27	18	28	4	2	10	28	0.474	0.226	0.180	110%	20%
s382	48	66	6	5	15	39	0.169	0.135	0.126	25%	7%
s526	56	77	6	3	21	46	0.222	0.181	0.176	23%	3%
s400	54	74	8	7	38	60	0.692	0.268	0.257	158%	4%
s444	70	94	12	5	33	45	0.656	0.269	0.236	144%	12%
s386	59	142	11	3	53	129	0.250	0.199	0.190	26%	5%
s1488	180	619	47	7	139	340	0.233	0.181	0.164	29%	9%

variable-delay units, the average delay is used as the delay of the transition. The average errors are 26% and 7% respectively. If both early evaluation and variable-delay nodes are taken into account, the average errors are 63% and 9%.

The LP solver used to compute the upper bound method is CPLEX. The presented method has been implemented in C++. The experiments have been run on a Xeon processor at 2.66 GHz with 3 MB of cache. Each simulation was performed for 10000 cycles. The lower bound method is around half to one order of magnitude slower than the LP method. Simulations were around 2 orders of magnitude slower than the lower bound method. The lower bound run for the biggest graph took around 5 seconds. Most of the other runs took less than one second.

B. Evaluating Relative Performance

If a designer wants to compare several possible designs or perform some architectural exploration, it is important that the relative order between designs on simulation and on analysis is the same.

We have modified each graph by adding and removing unit delays, which corresponds to adding and removing empty buffers, a correct transform in elastic systems[12]. After generating a set of 6 graphs from the same original graph, the ordering obtained by using the simulation throughput has been compared to the ordering obtained by using the two analysis methods. For designs with a difference of throughput higher than 4%, both the upper bound method and our method can correctly order the designs. For smaller differences in the throughput, none of the methods seems to be able to correctly discern the order.

VII. Conclusions

A method to analyze the performance of elastic systems with early evaluation and variable-latency units has been proposed. The method computes a lower bound of the throughput by unfolding a marked graph, deriving symbolic expressions for the firing times of the transitions and evaluating them. The accuracy of the lower bound can be incremented by increasing the number of correlations taken into account when the expressions are evaluated. The worst case running time when all correlations are taken into account is exponential.

Results show that an accurate lower bound of the throughput can be obtained with a time difference of around two orders of magnitude with regard to the simulation time.

A fast method to estimate the performance of an elastic system is important because it allows architectural exploration without running any simulations, which are too time consuming for an exploration framework.

Acknowledgments. This work has been supported by grants from Intel Corp., CICYT TIN2004-07925 and FI from Generalitat de Catalunya.

References

[1] A. Agarwal, D. Blaauw, V. Zolotov, and S. Vrudhula. Computation and refinement of statistical bounds on circuit delay. In *Proceedings of the 40th conference on Design automation*, pages 348–353. ACM New York, NY, USA, 2003.

[2] M. Akian, R. Bapat, and S. Gaubert. Max-plus algebra. *Handbook of Linear algebra: Discrete Mathematics and its Application, Chapman & Hall/CRC, Baton Rouge, LA*, 2007.

[3] A. Armando, C. Castellini, and E. Giunchiglia. Sat-based procedures for temporal reasoning. In *ECP '99: Proc. of the 5th European Conference on Planning*, pages 97–108, London, UK, 2000. Springer-Verlag.

[4] L. Benini, E. Macii, M. Poncino, and G. De Micheli. Telescopic units: a new paradigm for performance optimization ofVLSI designs. *IEEE Transactions on Computer-Aided Design*, 17(3):220–232, 1998.

[5] C. Brej. *Early Output Logic and Anti-Tokens*. PhD thesis, University of Manchester, 2005.

[6] R. Bryant. Binary decision diagrams and beyond: Enabling technologies for formal verification. In *Proc. ACM/IEEE Design Automation Conf.*, pages 236–243, 1995.

[7] J. Campos, G. Chiola, and M. Silva. Ergodicity and throughput bounds of Petri nets with unique consistent firing count vector. *IEEE Transactions on Software Engineering*, 17(2):117–125, 1991.

[8] L. Carloni, K. McMillan, and A. Sangiovanni-Vincentelli. Theory of latency-insensitive design. *IEEE Trans. on Computer-Aided Design*, 20(9):1059–1076, Sept. 2001.

[9] M. Casu and L. Macchiarulo. Adaptive Latency Insensitive Protocols and Elastic Circuits with Early Evaluation: A Comparative Analysis. *Electronic Notes in Theoretical Computer Science*, 245:35–50, 2009.

[10] J. Cortadella, M. Galceran-Oms, and M. Kishinevsky. Elastic Systems. In *Proc. 8th ACM/IEEE Int. Conf. on Formal Methods and Models for Codesign (MEMOCODE 2010)*, pages 149–158, July 2010.

[11] J. Cortadella and M. Kishinevsky. Synchronous elastic circuits with early evaluation and token counterflow. In *Proc. ACM/IEEE Design Automation Conf.*, pages 416–419, June 2007.

[12] J. Cortadella, M. Kishinevsky, and B. Grundmann. Synthesis of synchronous elastic architectures. In *Proc. ACM/IEEE Design Automation Conf.*, pages 657–662, July 2006.

[13] J. Esparza and K. Heljanko. *Unfoldings: a partial-order approach to model checking*. Springer-Verlag New York Inc, 2008.

[14] J. Gunawardena. Timing analysis of digital circuits and the theory of min-max functions. In *TAU'93, ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, 1993.

[15] H. M. Jacobson, P. N. Kudva, P. Bose, P. W. Cook, S. E. Schuster, E. G. Mercer, and C. J. Myers. Synchronous interlocked pipelines. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 3–12, Apr. 2002.

[16] J. Júlvez, J. Cortadella, and M. Kishinevsky. Performance analysis of concurrent systems with early evaluation. In *Proc. International Conf. Computer-Aided Design*, Nov. 2006.

[17] R. Karp. A characterization of the minimum mean-cycle in a digraph. *Discrete Maths*, 23:309–311, 1978.

[18] J. Liou, K. Cheng, S. Kundu, and A. Krstic. Fast statistical timing analysis by probabilistic event propagation. In *Proc. ACM/IEEE Design Automation Conf.*, pages 661–666. ACM, 2001.

[19] K. McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *Computer Aided Verification*, pages 164–177, 1992.

[20] T. Murata. Petri Nets: Properties, analysis and applications. *Proceedings of the IEEE*, pages 541–580, Apr. 1989.

[21] C. Nielsen and M. Kishinevsky. Performance analysis based on timing simulation. In *Proc. ACM/IEEE Design Automation Conf.*, pages 70–76. ACM, 1994.