Performance optimization of elastic systems using buffer resizing and buffer insertion

Dmitry Bufistov
Univ. Politècnica de Catalunya
Barcelona, Spain

Jorge Júlvez
Univ. Politècnica de Catalunya
Barcelona, Spain

Jordi Cortadella
Univ. Politècnica de Catalunya
Barcelona, Spain

***Abstract*—Buffer resizing and buffer insertion are two transformation techniques for the performance optimization of elastic systems. Different approaches for each technique have already been proposed in the literature. Both techniques increase the storage capacity and can potentially contribute to improve the throughput of the system. Each technique offers a different trade-off between area cost and latency. This paper presents a method that combines both techniques to achieve the maximum possible throughput while minimizing the cost of the implementation. The provided method is based on mixed integer linear programming. A set of experiments is designed to show the feasibility of the approach.**

I. Introduction

A. Elastic systems

Elastic systems (ES) offer a simple and elegant approach to variability tolerance in computation and communication delays. Thus, ESs may help to cope with the problem of long global interconnection delays that arises in contemporary nanotechnology [1].

Elasticity opens the door to a new set of circuit transformations that provide new opportunities for architectural exploration, trading-off area, delay and power. A key aspect of elastic systems is that they accept a set of valid transformations that preserve the circuit behavior regardless the timing characteristics of its components.

Traditionally, two main categories of circuits have been defined regarding the synchronization scheme of their components: asynchronous and synchronous systems. Elasticity is a general term that refers to both categories. In both cases, the components interact with pairs of handshake signals that exchange bidirectional information related to the validity of the data or the back-pressure of the pipelines.

In asynchronous circuits, the handshake signals are usually called *request* and *acknowledge*. In synchronous circuits these signals are usually called *valid* and *stop*[1]. A specific theory for synchronous elastic circuits was initially presented in [2], coining the term *latency-insensitive systems*. The theory presented in this paper covers both, synchronous and asynchronous ESs, without making an explicit distinction between them.

An ES is guaranteed to work correctly regardless of the computation and communication delays of their components. A computational node produces new valid output data only when every input is valid and every output is ready to accept them.

[1]To be more precise, the stop and acknowledge signals have inverse semantics.

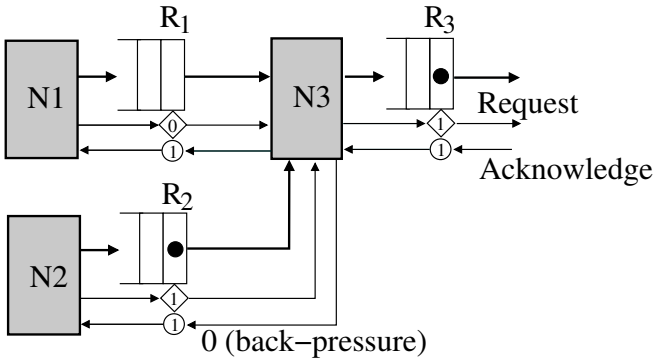


Figure 1. A simple elastic system.

Figure 1 shows an example of a simple ES with three computational nodes N1, N2 and N3, connected via *elastic buffers*. The value of the request signal is represented by the rhombus attached to each buffer. When the request signal is asserted, we say that the buffer stores *tokens* (valid data), otherwise it stores *bubbles* (non-valid data). If some of the inputs of a node are valid while others are not, the node must inform its inputs to keep the same valid data until all inputs become valid. For instance, node N3 in Fig. 1 must inform its input buffer R_2 to maintain the data until it has been consumed. In this case, R_2 is said to be stopped because of the *back-pressure* [3], [4]. The backward communication between nodes is implemented by the acknowledge signal. In Fig. 1, the value of the acknowledge signal is represented in the circle attached to each buffer.

This paper will not discuss the different handshake protocols for ESs, either asynchronous or synchronous. For specific implementations of these protocols, the reader can look at the specialized literature, e.g., [2], [5], [6].

B. Back-pressure and buffers

Back-pressure may be produced by several reasons:

- The environment is not ready to accept more tokens, thus stopping the ones coming from the system.
- The environment is not producing tokens and, thus, the nodes waiting for the tokens must stop the inputs already available in other channels.

978-1-4244-2820-5/08/$25.00 ©2008 IEEE

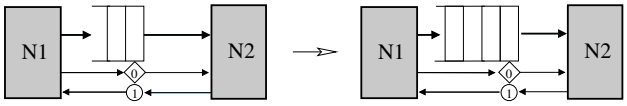


Figure 2. Buffer resizing: The capacity of the buffer is resized from 2 to 4.

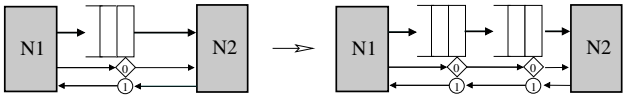


Figure 3. Buffer insertion: A new buffer with capacity 2 is inserted.

- Mismatches in the arrival time of tokens produced by the internal nodes to the inputs of *join* nodes.

Given that the behavior of the environment cannot be changed when designing a system, we focus our attention to reducing the performance losses coming from the third cause, i.e. mismatches in arrival times to join nodes.

This problem arises when the system has unbalanced fork-join paths. This is known as the *slack matching* problem [7], [8]. Buffer resizing (Fig 2) and insertion (Fig 3) are techniques aiming at balancing fork-join paths to improve system performance.

There are differences between both techniques that make necessary to explore different trade-offs to solve the slack matching problem. In a first simplistic view, buffer resizing increases the capacity of the buffer but not its latency. Buffer insertion increases the capacity and the latency.

However, there are subtle aspects that must be taken into account. The latency of a buffer is not independent of its size. Even if in the discrete world of synchronous systems it is often possible to make the simplistic assumption that all buffers have a 1-cycle delay, it is more realistic to assume that the latency of a buffer may slightly increase with its capacity. This aspect will be taken into account in our models.

On the other hand, buffer insertion implies an explicit increase of latency without modifying the cycle period. Buffer insertion is also more modular since it does not require to modify existing buffers in the system and requires a simpler control logic.

For the previous reasons, buffer insertion is usually preferred to mitigate the slack matching problem. However, buffer resizing will be chosen when necessary.

C. Modeling elastic systems

Several performance analysis of ESs can be carried out using *marked graph* [9] models which are a subclass of *Petri Nets* [10]. In this paper ESs are modeled as a particular class of marked graphs called *back-pressure graph* (BPG) [11].

Figure 4(a) shows the BPG of a simple ES. Each computational block is represented as a pair of forward and backward edges, e.g., (a, b) and (b, a). Forward edges are depicted as solid lines, backward edges are depicted as dashed lines. The forward edge delay represents the computational delay of the block. The backward edge delay represents the *acknowledgement* delay of the block. Each edge has associated a nonnegative delay. Assume that all delays are equal to one, i.e., it is a synchronous elastic system.

The vertices of the BPG represent fork and join controls. We assume that their delays are included in the forward and the backward delays of computational blocks.

Each forward edge has a token (marked with a solid point) if its corresponding block has a valid data. The number of tokens in backward edges represents the number of free registers in the corresponding buffer. Thus, block (d, a) is initialized with one valid data and has one free register. The block (a, b) has no valid data. The total number of tokens in the forward and backward edges of two adjacent nodes is the *capacity* of the buffer represented by such edges. The capacity of all buffers in Fig. 4(a) is 2.

D. Performance of elastic systems

An interesting performance measure of an ESs is its *throughput*. The throughput of a ES is the average number of tokens produced per time unit. The throughput of an ES is bounded by two quantities: the minimum cycle ratio [12], [13] and the maximum delay of the edges.

The minimum cycle ratio (mcr) of a BPG is defined as: $mcr = \min_{a \in A} \frac{\sum_{e \in a} T(e)}{\sum_{e \in a} \delta(e)}$ where A is the set of directed cycles of the BPG, $T(e)$ and $\delta(e)$ are the number of tokens and delay of edge e respectively.

Since a given edge e can produce valid data at most every $\delta(e)$ time units, the throughput of the ES is upper bounded by $1/\delta_{max}$ where $\delta_{max} = \max_{e \in E} \delta(e)$ where E is the set of edges of the BPG. Thus, the throughput of an ES is equal to $\min\left\{mcr, \frac{1}{\delta_{max}}\right\}$ [14], [15]. For example, the throughput of the BPG in Fig. 4(a) if all delays are 1 is equal to $\frac{2}{3}$, and the cycle giving the mcr is (a, b, c) (it has 2 tokens and the sum of delays is 3).

E. Buffer resizing

A usual technique to enhance the throughput of an ES is to increase the buffer capacities (see Fig. 2). Let us, for instance, resize the buffer of the input (a, c) in Fig. 4(a) from 2 to 3. This leads to the BPG depicted in Fig. 4(b) whose throughput is $\frac{3}{4}$ with (a, b, c, d) providing the mcr.

F. Buffer insertion

Another technique to enhance the throughput is to insert empty buffers, which are usually called bubbles, between computational nodes (see Fig. 3). The BPG in Fig. 4(c) is the result of inserting one buffer (a, r) in the BPG in Fig. 4(a). The throughput of the BPG in Figure 4(c) is equal to $\frac{3}{4}$ (the cycles giving the mcr are (a, b, c, d) and (a, r, c, d)) which is greater than the throughput of the original BPG ($\frac{2}{3}$).

G. Related work

Buffer resizing is a well known technique in synchronous latency insensitive design [4], while it has not been applied yet in asynchronous design. In [4] a mixed integer linear programming (MILP) based approach is proposed to maximize

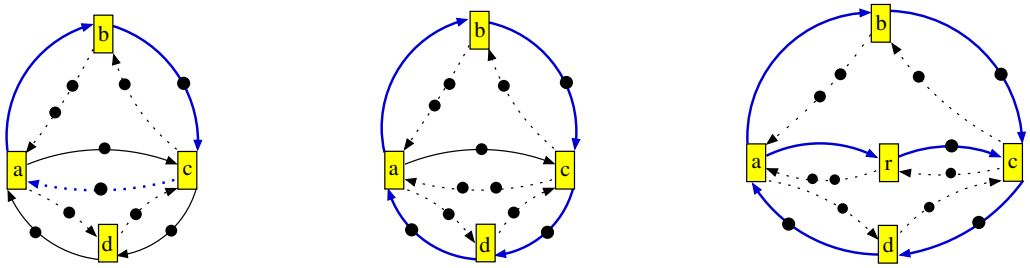


Figure 4. (a) A back-pressure graph; (b) The capacity of (a, c) is resized to 3; (c) One buffer is inserted in (a, c).

the throughput of latency insensitive systems by using buffer resizing.

Buffer insertion has a long history in the area of asynchronous design, where it is often called *slack matching*. In [16], it is shown that the performance of a self-timed ring is maximized when it is balanced with respect to the tokens to bubbles ratio, and a quantitative approach to calculate such an optimal balance is presented. In [15], a precise linear approximation for the performance analysis of iterative computations in self-timed rings is presented. The model is based on event-rule systems and it is closely related to previous models based on Petri nets [14], [17]. In [18], an iterative algorithm for buffer insertion in asynchronous systems is presented. In [7], [8], algorithms for buffer insertion in choice-free asynchronous systems are proposed, the algorithms are based on MILP models.

In [19], a polynomial time heuristics for buffer resizing was proposed. Moreover, it was observed that the throughput achievable with the simultaneous use of buffer resizing and buffer insertion may be greater than the throughput achievable only with buffer insertion.

H. Motivation and Contribution

Buffer insertion only requires the insertion of empty buffers. Such buffers can be efficiently implemented using latches [6]. The main drawback of buffer insertion is that it does not always achieve the same throughput as buffer resizing [19]. A simple BPG that demonstrates this limitation is shown in Fig. 5: If one token is added to the edge (d, a), i.e., the capacity of edge (a, d) is increased to 3, the system throughput becomes $\frac{4}{5}$ what cannot be achieved by any possible buffer insertion. The model presented in this paper automatically detects when buffer resizing is necessary to achieve the maximum throughput.

Unfortunately, the usage of buffers with non-minimal sizes (which are required after buffer resizing) increases the complexity of the control logic and consequently its computational delay.

The main contribution of this paper is a method that combines buffer insertion and buffer resizing for the performance optimization of ESs. In order to model more realistically ESs, the proposed method takes into account that the resizing of buffers may entail an increase of the system delays. The solution of the method achieves the maximum possible throughput and minimizes the overall cost (area, complexity, etc) of the implementation.

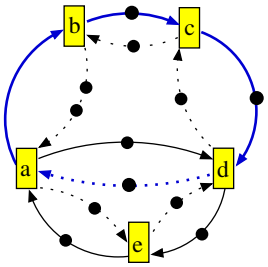


Figure 5. A BPG for which buffer resizing achieves a better throughput than buffer insertion.

II. Background

This section formalizes the notion of BPG and reviews a linear programming (LP) formulation to compute the throughput of a given BPG.

A. Back-pressure graph

Definition 2.1 (BPG): A back-pressure graph (BPG) is a directed connected bi-weighted graph $B = (V, E, E', T, \delta, C_0)$, where:

- V is the set of vertices.
- E is the set of forward edges.
- E' is the set of backward edges.
- $T : E \rightarrow \mathbb{N}$ is the initial number of tokens in each forward edge.
- $\delta : E \cup E' \rightarrow \mathbb{R}^+$ is the delay of each edge.
- $C_0 : E \rightarrow \mathbb{N}$ is the buffer capacity of e, where $C_0(e) \geq 2$ for every $e \in E$.

A bijection, $G : E \rightarrow E'$, exists such that for each forward edge $e = (u, v)$ its corresponding backward edge $e' = G(e)$ verifies $e' = (v, u)$. The number of tokens in e' is $C_0(e) - T(e)$ and represents the number of free registers in the corresponding buffer.

B. Throughput and minimum cycle ratio

The throughput, Θ, of a BPG is the number of tokens produced per time unit. The throughput of a given BPG can be computed using the minimum cycle ratio (tokens to delays) of the BPG [14], [17]:

Proposition 2.1: Let $B = (V, E, E', T, \delta, C_0)$ be a BPG. The throughput, Θ, of B is:

$$\Theta = \min\left\{ \min_{a \in A} \frac{\sum_{e \in a} T(e)}{\sum_{e \in a} \delta(e)}, \frac{1}{\delta_{max}} \right\} \quad (1)$$

where A is the set of directed cycles of B and $\delta_{max} = \max_{e \in E \cup E'} \delta(e)$.

A cycle a is called *critical* if it satisfies $\Theta = \frac{\sum_{e \in a} T(e)}{\sum_{e \in a} \delta(e)}$. A critical cycle can contain both forward and backward edges. If a critical cycle contains backward edges the throughput of the system is being constrained by the buffer capacities of the system, and therefore, it can be improved with buffer resizing and buffer insertion.

The maximal throughput, Θ^*, that can be achieved by buffer resizing and buffer insertion for a given BPG is bounded by the throughput, Θ^F, of the corresponding "forward" graph [11]:

Proposition 2.2: Let $B = (V, E, E', T, \delta, C_0)$ be a BPG. The maximal throughput, Θ^*, that can be achieved by buffer resizing and buffer insertion satisfies:

$$\Theta^* \leq \Theta^F = \min\left\{ \min_{a \in A^F} \frac{\sum_{e \in a} T(e)}{\sum_{e \in a} \delta(e)}, \frac{1}{\delta_{max}} \right\} \quad (2)$$

where A^F is the set of directed cycles containing only forward edges of B and $\delta_{max} = \max_{e \in E \cup E'} \delta(e)$.

Let us now review an LP formulation for the throughput which is the base of the proposed performance models.

Theorem 2.1: Let $B = (V, E, E', T, \delta, C_0)$ be a BPG. The throughput of B is at least Θ_g iff $\Theta_g \leq \frac{1}{\delta_{max}}$ and a function $r : V \rightarrow \mathbb{R}$ exists such that:

$$\begin{aligned} &T(e) - \Theta_g \cdot \delta(e) \geq r(v) - r(u), \\ &C_0(e) - T(e) - \Theta_g \cdot \delta(e') \geq r(u) - r(v) \ \forall\ e = (u, v) \in E \end{aligned} \quad (3)$$

where the first constraint applies to each forward edge $(T(e), \delta(e))$, and the second constraint applies to the corresponding backward edge e' $(C_0(e) - T(e), \delta(e'))$.

Proof: The proof is very similar to the proof of *Theorem 1 in [4]*. Let $T_0(e)$ be the number of tokens in edge $e \in E \cup E'$ of B.

($\Rightarrow$) Assume that for given constant $\Theta_g \leq \frac{1}{\delta_{max}}$ there is a function r s.t. (3) holds. Then for each cycle c:

$$\Sigma_{e \in c}(T_0(e) - \Theta_g \cdot \delta(e)) \geq \Sigma_{e \in c}(r(v) - r(u)) = 0,$$

Thus, $\frac{\sum_{e \in c} T_0(e)}{\sum_{e \in c} \delta(e)} \geq \Theta_g$, and, by Proposition (2.1), the throughput of B is at least Θ_g.

($\Leftarrow$) Let us assume that $\Theta \geq \Theta_g$, and let us construct a function r such that (3) holds.

By Proposition 2.1, it holds that $\Theta_g \leq mcr(B)$ where $mcr(B)$ is the minimal tokens to delay ratio of B.

Let $G = (V_0, E_0, W)$ be a weighted directed graph, where $V_0 = \{s\} \cup V, E_0 = E \cup E' \cup \{(s, v) | v \in V\}$,

$$W(e) = \begin{cases} 0, & e = (s, v), \\ T_0(e) - \Theta_g \cdot \delta(e), & \text{otherwise.} \end{cases}$$

The graph G has the same set of directed cycles as B. The weight of each directed cycle c of G is equal to $\Sigma_{e \in c}(T_0(e) - \Theta_g \cdot \delta(e)) \geq 0$. Therefore, the shortest path problem is well defined for G.

Let $p(s, v)$ be a weight of the shortest path in G from s to v, then set $r(v) = p(s, v)$. There is a path from s to v which consist from shortest path from s to u and edge e. Thus, for each edge $e = (u, v)$ the following chain of inequalities holds: $p(s, u) + W(e) = r(u) + T_0(e) - \Theta_g \cdot \delta(e) \geq p(s, v) = r(v)$ or equivalently $T_0(e) - \Theta_g \cdot \delta(e) \geq r(v) - r(u)$. ■

By Theorem 2.1, the throughput of a BPG is given by the solution of the following LP:

$$\begin{aligned} &maximize:\ \Theta, \\ &\text{subject to:} \\ &\quad T(e) - \Theta \cdot \delta(e) \geq r(v) - r(u), \\ &\quad C_0(e) - T(e) - \Theta \cdot \delta(e') \geq r(u) - r(v) \\ &\quad \Theta \leq \frac{1}{\delta(e)}, \quad \Theta \leq \frac{1}{\delta(e')} \quad \forall\ e = (u, v) \in E \end{aligned} \quad (4)$$

Efficient solutions exist for LP (4) [20], [21], as well as other efficient algorithms for the minimum cycle ratio problem [13]. A remarkable feature of the constraints used in LP (4) is that they allow us to combine buffer insertion and buffer resizing in a single MILP.

III. MILP FORMULATIONS

Firstly this section reviews MILP formulations for buffer resizing and buffer insertion. Then, a single MILP formulation is proposed to combine both techniques. Finally, a method based on a binary search is described to search for the maximum achievable throughput.

A. MILP: Buffer resizing

The capacity of a buffer e in a BPG is determined by $C_0(e)$ which is equal to the sum of tokens in e plus the tokens in e'. An increase of the capacity of e is expressed in the BPG as an increase of the number of tokens in e'. For instance, the buffer in Fig. 6(a) has capacity 2 and the addition of 1 token to $e' = (v, u)$ increases its capacity to 3, see Fig. 6(b). The capacity of a buffer e after resizing is denoted by $C(e)$.

The following model for buffer resizing of ESs takes into account that an increase of a given buffer capacity entails a more complex control logic, and in turn an increase of the corresponding edge delay. Such an increase of the delay is usually logarithmic with respect to the increase of the capacity, however, linear approximations work also reasonably well for small values of the capacity. Let us assume that the buffer corresponding to edge e is resize from capacity $C_0(e)$ to $C(e)$, then its new delay, $\delta_c(e)$, is:

(a) (b)

Figure 6. (a) Original buffer with capacity 2, i.e., $C_0(e) = 2$; (b) Buffer is resized to capacity 3, i.e., $C(e) = 3$.

$$\delta_c(e) = \delta(e) + h \cdot (C(e) - C_0(e)) \tag{5}$$

where h is a real parameter to adjust the linear approximation of the increase of delay with respect to the increase of capacity. Obviously, if $h = 0$ the new delay is independent of the new capacity.

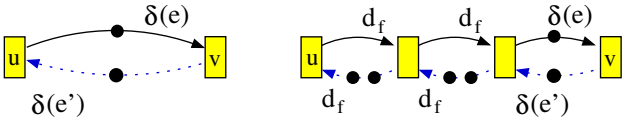
Capacity dependent delays, δ_c, can be included in Theorem 2.1 just by replacing $\delta(e)$ with $\delta_c(e)$. For instance, for a given $\Theta_g \leq \frac{1}{\delta_{max}}$ the minimum sum of capacities that can achieve Θ_g can be obtained by solving the following MILP:

$$\begin{aligned} &minimize: \sum_{e \in E} C(e), \\ &\text{subject to:} \\ &\quad T(e) - \Theta_g \cdot \delta_c(e) \geq r(v) - r(u), \\ &\quad C(e) - T(e) - \Theta_g \cdot \delta_c(e') \geq r(u) - r(v), \\ &\quad C(e) \in \mathbb{N}, C(e) \geq C_0(e) \text{ for each } e = (u,v) \in E \end{aligned} \tag{6}$$

where $C_0(e)$ is the initial buffer capacity of the edge e, and $C(e)$ is a variable determining the new capacity of e. The MILP (6) is feasible iff Θ_g is achievable by buffer resizing. If all delays in (6) are equal to one and $h = 0$ then (6) is equivalent to the model for buffer resizing of latency insensitive systems proposed in [4].

B. MILP: Buffer insertion

In order to describe buffer insertion of a BPG one more edge weight function is introduced: $N : E \rightarrow \mathbb{N}$, that specifies the number of empty pipelined buffers, i.e., bubbles, of each edge. For example, all the edges of the BPG in Fig. 4(c) have no pipelined buffers except (a, c), which is pipelined with one empty buffer (a, r), i.e., $N((a, c)) = 1$. Figure 7 exemplifies how the BPG is modified when 2 empty buffers are inserted.



(a) (b)

Figure 7. (a) Original buffer with capacity 2; (b) Result of inserting 2 empty buffers, i.e., $N(e) = 2$, with capacity 2.

After buffer insertion, the number of tokens on the backward edges from v to u is $C(e) - T(e) + C_b \cdot N(e)$, where C_b is the capacity of the inserted buffers. For example, $C_b = 2$ for latency insensitive systems. Let d_f and d_b be the forward and backward delays of the inserted empty buffers. Then, the overall delay of the forward edges from u to v is $\delta(e) + d_f \cdot N(e)$, and the overall delay of the backward edges from v to u is $\delta(e') + d_b \cdot N(e)$.

The use of Theorem 2.1 allows us to design a MILP that minimizes the number of buffers that must be inserted to achieve a given throughput $\Theta_g \leq \frac{1}{\delta_{max}}$:

$$\begin{aligned} &minimize: \sum_{e \in E} N(e) \\ &\text{subject to:} \\ &\quad T(e) - \Theta_g \cdot (\delta(e) + d_f \cdot N(e)) \geq r(v) - r(u), \\ &\quad C_0(e) - T(e) + C_b \cdot N(e) \\ &\qquad\qquad -\Theta_g \cdot (\delta(e') + d_b \cdot N(e)) \geq r(u) - r(v), \\ &\quad N(e) \in \mathbb{N}, \text{ for each } e = (u,v) \in E \end{aligned} \tag{7}$$

where $N(e)$ is the number of empty buffers inserted in edge e (capacity dependent delays, δ_c, are not considered in (7) because capacities are kept to their initial values C_0). The MILP (7) is feasible iff Θ_g is achievable by buffer insertion. The formulation in (7) is equivalent to the one obtained in [7] for slack matching in asynchronous design.

C. MILP: buffer resizing and buffer insertion

The described transformations techniques have their own advantages and disadvantages: buffer resizing might achieve a higher throughput than buffer insertion but the complexity of the required logic increases with the size of the computed capacities; buffer insertion does not increase the complexity of the control logic but it might achieve a lower throughput. The MILP proposed in this section aims at combining both techniques to minimize the cost involved by the implementation of the solution. The cost can refer to any index related to the resizing and insertion of buffers, e.g., area required to resize a buffer (insert a new buffer), complexity of the control logic required to resize a buffer (insert a new buffer), etc.

In order to combine both techniques, we will make use of a real parameter $\alpha > 0$ that represents the ratio of the cost required to increase in one unit the capacity of a buffer to the cost required to insert a bubble between two nodes. If $\alpha = 1 (\alpha < 1)(\alpha > 1)$ the cost required to increase in one the buffer size is assumed to be equal to(less than)(greater than) the cost to insert one bubble. For instance, if one desires to minimize the area of the implementation, and the area required to insert one bubble is half the area required to increase the capacity of a buffer in one unit, then α must be set to 2. For a given $\Theta_g \leq \frac{1}{\delta_{max}}$ the solution of the following MILP provides simultaneously a resizing and an insertion transformation that achieves Θ_g and minimizes the overall cost.

$$\begin{aligned} &minimize: \alpha \cdot \sum_{e \in E} C(e) + \sum_{e \in E} N(e) \\ &\text{subject to:} \\ &\quad T(e) - \Theta_g \cdot (\delta_c(e) + d_f \cdot N(e)) \geq r(v) - r(u), \\ &\quad C(e) - T(e) + C_b \cdot N(e) \\ &\qquad\qquad -\Theta_g \cdot (\delta_c(e') + d_b \cdot N(e)) \geq r(u) - r(v), \\ &\quad N(e), C(e) \in \mathbb{N}, C(e) \geq C_0 \text{ for each } e = (u,v) \in E \end{aligned} \tag{8}$$

where $C(e)$ is the capacity of edge e, and $N(e)$ is the number of empty buffers inserted in edge e. The MILP (8) is feasible iff Θ_g is achievable by buffer resizing and buffer insertion. Notice that even a very high α is chosen, it might not be possible to achieve Θ_g just by inserting bubbles (recall Subsection I-H).

D. Search for the maximum throughput

The use of MILP (8) greatly eases the search for the maximum throughput, Θ^*, achievable with buffer resizing and buffer insertion. By Proposition 2.2, Θ^* is upper-bounded by Θ^F, i.e., $\Theta^* \leq \Theta^F$. On the other hand, Θ^* is obviously greater than or equal to the throughput, Θ, of the original BPG which is given by (1), then $\Theta \leq \Theta^* \leq \Theta^F$.

This way, the interval $[\Theta, \Theta^F]$ can be taken to perform a binary search for Θ^*: if (8) is feasible for a given $\Theta_1 \in [\Theta, \Theta^F]$ then $\Theta_1 \leq \Theta^* \leq \Theta^F$ and the interval $[\Theta_1, \Theta^F]$ is taken for the next search; if (8) is not feasible for Θ_1 then $\Theta \leq \Theta^* \leq \Theta_1$ and the interval to be considered is $[\Theta, \Theta_1]$. This procedure can be repeated until a satisfactory precision for Θ^* is obtained.

IV. Experimental results

A. Generation of BPGs

Two sets of BPGs have been used to evaluate the method presented in this paper. On the one hand, the ISCAS89 circuits have been used to extract the underlying graph. On the other hand, some random graphs have also been generated (from ge1 to ge10). The tokens, buffer capacities and delays of the BPGs have been generated as follows:

Initial tokens: Each forward edge e is assigned a token, i.e., $T(e) = 1$, with probability 0.9.

Capacities: The initial capacity of each edge e is 2, i.e., $C_0(e) = 2$. The capacity of the inserted buffers is 2 as well, i.e., $C_b = 2$.

Delays: Each forward delay e is generated randomly, more precisely, $\delta(e)$ is a real number obtained from a random uniform variable in the interval $[1,\ 3]$. The forward delay of the inserted buffers is 1, i.e., $d_f = 1$. All backward delays are set to 1, i.e., $\delta(e') = 1$ for each backward edge e', and $d_b = 1$ for the inserted buffers. The dependence of the delay on the capacity is modeled using Equation (5) with $h = 0.5$.

In order to test buffer resizing and buffer insertion, every BPG in Table I satisfies $\Theta < \Theta^F$, what implies that its throughput can be improved.

B. Results

For each test case, buffer resizing and buffer insertion solutions that provide a maximal throughput were obtained by using the procedure in Subsection III-D. Three different values of α were used, $6/5$, 1, and $5/6$, in order to test different priorities for buffer resizing and buffer insertion.

Table I reports the obtained results. Columns $|V|$ and $|E|$ are the number of vertices and forward edges respectively. Column "Θ" is the throughput of the BPG computed with (1). Column "Θ^*" provides the maximal throughput achieved with buffer resizing and buffer insertion. Column "ΣN" reports the number of inserted buffers. Column "$\Sigma\Delta C$" reports the overall increase of buffer capacities, $\sum_{e \in E} (C(e) - C_0(e))$.

Observation 1: In every case, except ge5 and ge6, the throughput upper bound, Θ^F, provided by (2) is achievable, i.e., $\Theta^* = \Theta^F$. For ge5 $\Theta^F = 0.4037$ and $\Theta^* = 0.4025$; for ge6 $\Theta^F = 0.4653$ and $\Theta^* = 0.4643$, the precision of Θ^* obtained by the binary search was set to 0.01%. That is, the increase of delays due to buffer resizing avoids achieving Θ^F only in ge5 and ge6 (notice, however, that $\Theta^F - \Theta^*$ is very small in both cases) .

Observation 2: In most cases Θ^* can be achieved only by buffer insertion, see columns "ΣN" and "$\Sigma\Delta C$" below $\alpha = 6/5$. Only ge1, ge2, ge3, and ge4 require buffer resizing to achieve Θ^* (this happens even if arbitrarily high values of α are considered).

Observation 3: With $\alpha = 5/6$, Θ^* is achieved only by buffer resizing in all cases except in s27, s400 and ge7. Nevertheless, if α is set to 0.5, i.e., higher cost is given to buffer insertion, the maximum throughput of s27, s400 and ge7 is achieved only by buffer resizing with $\Sigma\Delta C$ equal to 12, 9 and 10 respectively.

Observation 4: In all cases except s27, s400 and ge7, it holds that $\Sigma\Delta C + \Sigma N$ keeps the same for any $\alpha > 0$ (see observation 3). In all cases except ge1, ge2, ge3, and ge4, the maximum throughput can be achieved with buffer insertion only (see observation 2). Therefore, in all cases except s27, s400, ge1, ge2, ge3, ge4, and ge7, if $\alpha = 1$ then any interchange of new buffers and capacities achieves Θ^* with the same cost in (8).

Observation 5: Although MILP problems are NP-complete, the CPU times spent for the largest test cases ge7, ge8, ge9 and ge10 were 10, 15, 45 and 540 seconds respectively. This because only the variables corresponding to cycles with minimum cycle ratio close to Θ are significant in the objective function, the rest of variables are set to 0 in the first steps performed by the solver. The CPU time for the rest of cases was less than one second.

All the experiments were run on XEON 3.8Gh with 14Gb RAM. CPLEX [22] was used as MILP solver.

V. Conclusions

The performance of an elastic system can be enhanced by using appropriate buffers between computational blocks. The main system transformations concerning buffers are the resizing of existing buffers and the insertion of new empty buffers. Buffer resizing might achieve higher throughput than buffer insertion, but its implementation is usually more complex than the insertion of new buffers.

In order to exploit the advantages of both transformations, a method that combines buffer resizing and buffer insertion has been proposed. The solution of the method provides the maximum throughput achievable with buffer resizing and buffer insertion, and minimizes the cost involved by the implementation of the resized and new buffers. The model used along the paper takes into account that the system delays depend on the capacity of the buffers. As it is shown in the experimental results, the method is applicable to large systems.

Table I
EXPERIMENTAL RESULTS

	$\|V\|$	$\|E\|$	Θ	Θ^*	Buffer resizing & buffer insertion					
					$\alpha = 6/5$		$\alpha = 1$		$\alpha = 5/6$	
					ΣN	$\Sigma\Delta C$	ΣN	$\Sigma\Delta C$	ΣN	$\Sigma\Delta C$
s27	31	78	0.2634	0.3399	11	0	11	0	1	10
s298	823	7154	0.0341	0.0383	37	0	37	0	0	37
s349	139	241	0.2110	0.2663	2	0	2	0	0	2
s400	119	273	0.1963	0.2892	8	0	8	0	1	7
s526	145	382	0.1215	0.1835	2	0	2	0	0	2
s641	182	298	0.2062	0.2855	2	0	2	0	0	2
s713	208	350	0.2935	0.3099	1	0	1	0	0	1
s820	183	919	0.1094	0.1320	10	0	10	0	0	10
s832	191	972	0.1181	0.1356	7	0	7	0	0	7
s953	373	704	0.2887	0.3233	15	0	15	0	0	15
s1423	484	942	0.1529	0.1784	1	0	1	0	0	1
s1488	321	1662	0.0716	0.0867	1	0	1	0	0	1
s1494	341	1775	0.0639	0.0911	22	0	22	0	0	22
s5378	1138	2484	0.2204	0.2527	5	0	5	0	0	5
s9234	1023	1992	0.1813	0.2133	3	0	3	0	0	3
ge1	10	30	0.5288	0.5897	3	1	3	1	0	4
ge2	20	50	0.5484	0.5916	2	1	2	1	0	3
ge3	70	100	0.5467	0.5785	2	1	2	1	0	3
ge4	30	100	0.5317	0.6470	6	1	6	1	0	7
ge5	20	100	0.3373	0.4025	1	0	1	0	0	1
ge6	10	30	0.4390	0.4642	1	0	1	0	0	1
ge7	5000	20000	0.0942	0.1250	9	0	9	0	1	8
ge8	10000	50000	0.0662	0.0834	8	0	8	0	0	8
ge9	20000	100000	0.0675	0.0857	10	0	10	0	0	10
ge10	50000	500000	0.0146	0.0220	8	0	8	0	0	8

Acknowledgements. This research has been funded by a grant from Intel Corp., research projet CICYT TIN2007-66523, FPU grant AP2005-4866, and a Juan de la Cierva fellowship from the Spanish Ministry of Education and Science.

REFERENCES

[1] J. Cong, "Challenges and opportunities for design innovations in nanometer technologies. in SRC working paper, Dec." 1997.

[2] L. P. Carloni, K. L. McMillan, A. Saldanha, and A. L. Sangiovanni-Vincentelli, "A methodology for correct-by-construction latency-insensitive design," in *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 1999, pp. 309–315.

[3] L. P. Carloni and A. L. Sangiovanni-Vincentelli, "Coping with latency in SoC design," *IEEE Micro, Special Issue on Systems on Chip*, vol. 22, no. 5, p. 12, October 2002.

[4] R. Lu and C.-K. Koh, "Performance optimization of latency insensitive systems through buffer queue sizing of communication channels," in *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 2003, pp. 227–231.

[5] J. Sparsø and S. Furber, Eds., *Principles of Asynchronous Circuit Design: A Systems Perspective*. Kluwer Academic Publishers, 2001.

[6] J. Cortadella, M. Kishinevsky, and B. Grundmann, "Synthesis of synchronous elastic architectures," in *Proc. ACM/IEEE Design Automation Conference*, Jul. 2006, pp. 657–662.

[7] P. A. Beerel, N.-H. Kim, A. Lines, and M. Davies, "Slack matching asynchronous designs," in *Proc. of the 12th Int. Symp. on Asynchronous Circuits and Systems*, 2006.

[8] P. Prakash and A. J. Martin, "Slack matching quasi delay-insensitive circuits," in *Proc. of the 12th Int. Symp. on Asynchronous Circuits and Systems*, 2006.

[9] F. Commoner, A. W. Holt, S. Even, and A. Pnueli, "Marked directed graphs," *Journal of Computer and System Sciences*, vol. 5, pp. 511–523, 1971.

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

[11] R. Lu and C.-K. Koh, "Performance analysis of latency-insensitive systems." *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 25, no. 3, 2006.

[12] R. Karp, "A characterization of the minimum cycle mean in a digraph," *Discrete Mathematics*, vol. 23, pp. 309–311, 1978.

[13] A. Dasdan, S. Irani, and R. Gupta, "Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems," in *Proc. 36th Design Automation Conference*, 1999, pp. 37–42.

[14] C. Ramchandani, "Analysis of asynchronous concurrent systems by timed Petri nets," Massachusetts Inst. of Tech., Tech. Rep. Project MAC Tech. Rep. 120, Feb. 1974.

[15] S. M. Burns and A. J. Martin, "Performance analysis and optimization of asynchronous circuits," in *Advanced Research in VLSI*. MIT Press, 1991, pp. 71–86.

[16] T. E. Williams, "Performance of iterative computation in self-timed rings," *Journal of VLSI Signal Processing*, vol. 7, no. 1/2, pp. 17–31, Feb. 1994.

[17] C. V. Ramamoorthy and G. S. Ho, "Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets," *IEEE Trans. on Software Engineering*, vol. 6, no. 5, pp. 440–449, 1980.

[18] K. Fazel, L. Li, M. Thornton, R. B. Reese, and C. Traver, "Performance enhancement in phased logic circuits using automatic slack-matching buffer insertion," in *GLSVLSI '04: Proceedings of the 14th ACM Great Lakes symposium on VLSI*. New York, NY, USA: ACM, 2004, pp. 413–416.

[19] R. Collins and L. Carloni, "Topology-based optimization of maximal sustainable throughput in a latency-insensitive system," in *The Proceedings of the Design Automation Conference*, June 2007, pp. 410–416.

[20] S. M. Burns, "Performance analysis and optimization of asynchronous circuits," Ph.D. dissertation, California Institute of Technology, 1991.

[21] J. Cochet-Terrasson, G. Cohen, S. Gaubert, M. M. Gettrick, and J.-P. Quadrat, "Numerical computation of spectral elements in max-plus algebra," *Proc. of the IFAC Conference on System Structure and Control*, july 1998.

[22] "CPLEX," Available from http://www.ilog.com.