Symbolic Analyses of Dataflow Graphs[1]

ADNAN BOUAKAZ, PASCAL FRADET, and ALAIN GIRAULT, Univ. Grenoble Alpes, Inria, CNRS, LIG, F-38000 Grenoble, France

The synchronous dataflow model of computation is widely used to design embedded stream-processing applications under strict quality-of-service requirements (*e.g.,* buffering size, throughput, input-output latency). The required analyses can either be performed at compile time (for design space exploration) or at run-time (for resource management and reconfigurable systems). However, these analyses have an exponential time complexity, which may cause a huge run-time overhead or make design space exploration unacceptably slow.

In this paper, we argue that *symbolic* analyses are more appropriate since they express the system performance as a function of parameters (*i.e.,* input and output rates, execution times). Such functions can be quickly evaluated for each different configuration or checked *w.r.t.* different quality-of-service requirements. We provide symbolic analyses for computing the maximal throughput of acyclic synchronous dataflow graphs, the minimum required buffers for which as soon as possible scheduling achieves this throughput, and finally the corresponding input-output latency of the graph. The paper first investigates these problems for a single parametric edge. The results are extended to general acyclic graphs using linear approximation techniques. We assess the proposed analyses experimentally on both synthetic and real benchmarks.

CCS Concepts:•**Theory of computationn $\rightarrow$ Models of computation;•Computing methodologies $\rightarrow$ Symbolic and algebraic algorithms;**

Additional Key Words and Phrases: Synchronous dataflow graphs, Static analysis, Throughput, Latency, Buffer minimization

ACM Reference Format:
Adnan Bouakaz, Pascal Fradet, and Alain Girault, 2016. Symbolic Analyses of Dataflow Graphs. *ACM Trans. Des. Autom. Electron. Syst.* V, N, Article 0000 (2016), 25 pages.
DOI: 0000001.0000001

1. INTRODUCTION

Embedded stream-processing applications become computationally intensive with strict quality-of-service requirements. Many-core platforms are hence required for performance, scalability and energy consumption reasons [Kumar et al. 2011]. To take advantage of such platforms, design models should express task-level parallelism and be simple enough to allow predictable system design.

Dataflow process networks (DPN) [Dennis 1974] and Kahn process networks (KPN) [Kahn 1974] allow to explicitly express parallelism and communications where tasks (or actors) are independent and communicate only through channels. Using a dataflow model of computation (MoC), concurrency can be implemented without explicit synchronization mechanisms and data races are ruled out by construction. Furthermore, these models are inherently functionally deterministic, *i.e.,* for the same

[1]This paper is an extended and improved version of [Bouakaz et al. 2016b].

Authors' address: Inria, France. Univ. Grenoble Alpes, Lab. LIG, Grenoble, France. CNRS, Lab. LIG, F-38000 Grenoble, France. Email: *first.last@inria.fr*

sequence of inputs, the system will always produce the same sequence of outputs. However, many important properties such as *boundedness* (*i.e.,* the system can execute in finite memory) and *liveness* (*i.e.,* no part of the system will deadlock) are undecidable.

The synchronous dataflow (SDF) model [Lee and Messerschmitt 1987] is a restriction of DPN and comes with static analyses that guarantee the *boundedness* and *liveness* of an application as well as *predictable performances* (*e.g.,* throughput, latency, memory requirements). For these reasons, it is widely used to design digital signal processing and concurrent real-time streaming applications on many-core platforms.

In response to the increasing complexity of stream-processing systems, many parametric extensions of the SDF model have been proposed (*e.g.,* PSDF [Bhattacharya and Bhattacharyya 2001], SPDF [Fradet et al. 2012], BPDF [Bebelis et al. 2013], πSDF [Desnos et al. 2013], *etc.*) in which the graph (*e.g.,* its communication rates or channels) may change at run-time.

Performance analyses of SDF graphs are used to check whether non-functional requirements are met. They can be performed both at design time and at run-time. At design time, it is a crucial step in the development of embedded applications. Many decisions and settings of the system need to be explored (e.g. hardware/software partitioning, memory allocation, granularity and different implementations of tasks, processor speeds, etc.) and the best options that satisfy the non-functional requirements can be chosen. At run-time, performance analysis is performed either for resource management or to cope with the dynamic behavior of parametric extensions of SDF.

The most prominent performance constraints of real-time stream-processing systems are throughput, latency and memory. Throughput is a crucial timing constraint of stream-processing systems. For example, a video decoder is supposed to decode a minimum number of frames per second. A throughput-optimal scheduling policy, such as self-timed scheduling, allows the designer to guarantee timing requirements. Latency is another important timing constraint that is usually used in the design of real-time control systems. It measures the time delay between stimulation and response, and hence the reactiveness of the system and its ability to react in a timely way. Finally, most embedded systems must comply to severe constraints on the size, weight, power and cost. Therefore, the minimization of memory requirements is a crucial step in the design of such systems. Throughput, latency, and memory measures are often antagonistic. Huge efforts have been devoted in the past decades to solve these problems.

We focus on self-timed scheduling that produces maximal throughput (with sufficiently large buffers). We propose *symbolic* analyses of dataflow graphs where communication rates and execution times of actors are *parameters*. Most non-functional properties of the application can be described as a function of these parameters. By evaluating these functions for specific values, the properties and performance of specific configurations can be obtained efficiently. We propose three symbolic analyses of acyclic graphs under self-timed scheduling to answer the following questions:

Q1. What is the *throughput* of the application?

Q2. What are the *minimum channel sizes* that allow maximum throughput?

Q3. What is the *latency* of the application under such channel sizes?

Although our symbolic analyses may give only approximate (but safe) results, they are very useful in many cases (see Fig. 1):

(i) At early design stages, the SDF graph modeling the application is only partially specified and design space exploration may require a potentially huge number of configurations to be analyzed (path $\boxed{3}$). Symbolic analyses are a big advantage in this case: formulas are generated only once and simply evaluated for each possible configuration

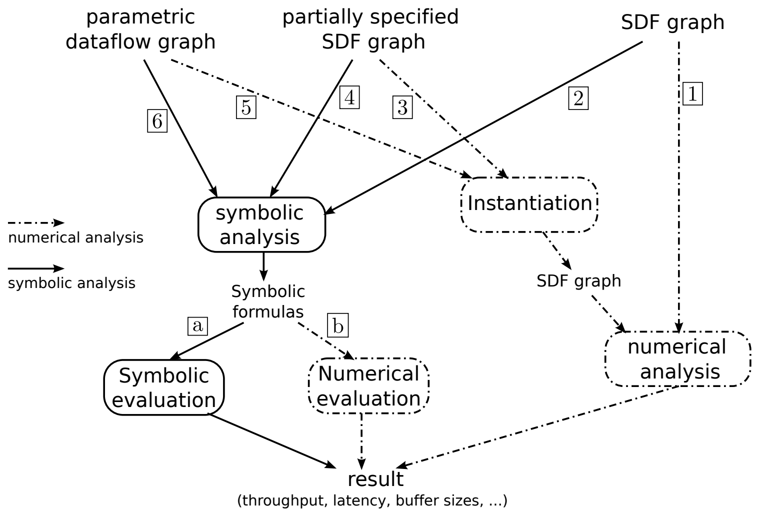


Fig. 1. Symbolic and numerical analyses.

(*i.e.,* set of parameters) (path $\boxed{4}$). Indeed, exact algorithms for throughput and latency computation may be acceptable at compile-time; however, frequent calls to these algorithms to check a large set of configuration values make design space exploration unacceptably slow.

(ii) Similarly, non-functional requirements of parametric dataflow models can be expressed symbolically as parametric formulas *at compile-time*. Then, the requirements can be either checked by *evaluating* formulas for all potential configurations (path $\boxed{6b}$) or, better, by an *analytic proof* (path $\boxed{6a}$). For instance, the designer could be interested in ensuring at compile-time that the throughput of the application is never below some given quality-of-service regardless of parameters changes at runtime.

(iii) For dynamic models and run-time resource management, appropriate settings have sometimes to be chosen dynamically. Consider a parametric application where frequency scaling is used to guarantee a specific throughput while minimizing power consumption: frequency must be adjusted at each parameter change. Instantiating the graph (path $\boxed{5}$) and performing a numerical analysis is far too costly at run-time. Consequently, fast analyses, like the evaluation of symbolic formulas, are required (path $\boxed{6b}$).

(iv) Finally, even for completely static SDF graphs, many analyses have an exponential complexity. Exact algorithms for minimal buffer sizes are too expensive even for small graphs: [Moreira et al. 2010] shows that this problem is NP-complete for homogeneous SDF (HSDF) graphs. Besides, SDF-to-HSDF conversion may lead to an *exponential* growth of the size of the graph. Our symbolic analysis (path $\boxed{2}$) is much more efficient and its approximate solution can also be considered as a starting point to prune the parameter space and hence improve the performance of the exact algorithm.

This paper is an extended and improved version of [Bouakaz et al. 2016b] which was limited to the symbolic computation of buffer sizes. Section 2 introduces the application model, the scheduling policy, and the definitions. Section 3 presents the throughput analysis of acyclic SDF graphs and the duality theorem required to solve the other questions. Section 4 presents different symbolic analyses for a simple SDF graph with a single edge $A \xrightarrow{p \quad q} B$ where p (resp. q) is the symbolic production (resp. consumption) rate of actor A (resp. B). Section 5 describes linearization techniques for graph $A \xrightarrow{p \quad q} B$ that are used in Sections 6 and 7 to extend the results of Section 4 to general acyclic graphs. Section 8 presents experiments conducted on synthetic and real-case benchmarks. Finally, we review related work in Section 9 and conclude in Section 10.

The interested reader will find a sketch of the main proofs in the electronic appendix and yet more additional details in a companion paper [Bouakaz et al. 2016a].

2. BACKGROUND

2.1. Application model

An SDF graph $G = (V, E)$ consists of a finite set of *actors* (computation nodes) V and a finite set of *edges* E that can be seen as unbounded FIFO channels. The execution of an actor (called *firing*) first consumes data tokens from all of its incoming edges (its *inputs*), then computes and finishes by producing data tokens to all of its outgoing edges (its *outputs*). The number of tokens consumed (resp. produced) at a given input (resp. output) edge at each firing is called its consumption (resp. production) *rate*. An actor can fire only when all its input edges have enough tokens, *i.e.,* at least the number specified by the corresponding rate. An edge may contain some *initial tokens*. Finally, we denote by t_X the execution time of actor X.

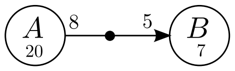


Fig. 2. The simple SDF graph with $t_A = 2$, $t_B = 7$, and one initial token.

For instance, Fig. 2 shows an SDF graph with two actors A and B, with execution times $t_A = 20$ and $t_B = 7$. The production and consumption rates on channel $A \rightarrow B$ are 8 and 5. This edge carries one initial token, represented by the black dot.

Each edge carries zero or more tokens at any moment. The *state* of a dataflow graph is the vector of the number of tokens present at each edge. The *initial state* of a graph is specified by the number of initial tokens on its edges. The initial state of the graph of Fig. 2 is represented by the vector $[i_{AB} = 1]$.

An *iteration* of an SDF graph is a non empty sequence of firings that returns the graph to its initial state. For the graph in Fig. 2, firing actor A five times (producing 40 tokens) and actor B eight times (consuming 40 tokens) forms an *iteration*. The *repetition vector* $\vec{z} = [z_A{=}5, z_B{=}8]$ indicates the number of firings of actors per iteration. If such a vector exists, then the graph is said to be *consistent* [Lee and Messerschmitt 1987]. We denote by z_X the number of firings of actor X in the iteration. The repetition vector is obtained by solving a system of *balance equations*. Each edge $A \xrightarrow{p \quad q} B$ is associated with the balance equation $z_A p = z_B q$, which states that all produced tokens during an iteration must be consumed within the same iteration.

Homogeneous SDF (HSDF) is a restriction of SDF where all the production and consumption rates are equal to 1. HSDF graphs are particularly useful because (*i*) any consistent SDF graph can be converted into an HSDF graph; and, (*ii*) the throughput of an HSDF graph can be computed as the inverse of the *Maximal Cycle Ratio* (MCR) of the graph (or *Maximum Cycle Mean*, MCM[2]). The ratio of a cycle is equal to the sum of execution times of the actors in the cycle divided by the number of initial tokens in the channels of this cycle. This provides a way to compute the throughput of any SDF graph. Yet, there are two important drawbacks: first, the translation from SDF

[2]Although the MCR and MCM differ slightly, they are often used indifferently in the dataflow literature.

to HSDF in general leads to an exponential increase of the number of nodes; second, partially specified or parametric SDF graphs cannot be converted into HSDF.

2.2. Scheduling policy

In this paper, we focus on as soon as possible (ASAP) scheduling of consistent graphs without auto-concurrency (*i.e.,* two firings of the same actor cannot overlap). In such self-timed executions [Sundararajan Sriram 2000], an actor fires as soon as it becomes idle (no auto-concurrency) and has enough tokens on its input channels. We assume that there are sufficient processing units, *e.g.,* there are as many processors as actors or all actors are implemented in hardware. ASAP scheduling allows the graph to reach its maximal throughput. Such schedules are naturally pipelined and composed of a *transient* phase followed by a *steady state* that repeats infinitely. Fig. 3 shows the ASAP schedule of a simple SDF graph: each rectangle is one actor firing whose length is proportional to the actor's execution time, and the thick broken lines mark the iterations boundaries.

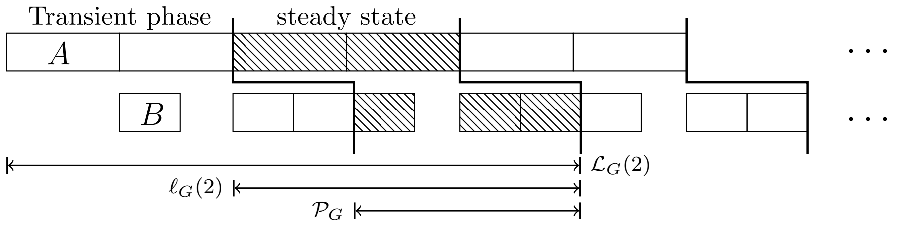


Fig. 3. ASAP schedule of the SDF graph $A \xrightarrow{3\quad 2} B$ with t_A=15 and t_B=8.

Fig. 4 illustrates how to make the absence of auto-concurrency explicit in an SDF graph by adding self-edges with rates equal to 1 and a single initial token: firing A consumes the unique token in its self-edge, preventing any other firing of A until another token is produced to the self-edge at the end of the current firing. Disabling auto-concurrency is mandatory for stateful actors to ensure proper state update.

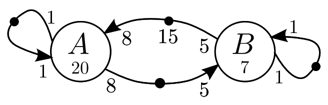


Fig. 4. The SDF graph of Fig. 2 with auto-concurrency disabled and channel size bounded by 15.

Channels are unbounded in SDF. However, the size of a channel $A \xrightarrow{p\quad q} B$ can be constrained to contain at most d tokens by adding a backward channel $B \xrightarrow{q\quad p} A$ with d initial tokens, as shown in Fig. 4. This modeling, assumed in most works, enforces that an actor can start firing only if there is enough space on its output channels. Moreover, the empty space is made available not at the beginning of the firing of the consumer but when it produces the tokens representing buffer places. One could imagine a less conservative modeling where the consumer makes the empty space available just after consumption, and the producer checks whether there is enough empty space only at the end of its firing. However, the approach proposed in this paper for symbolic computation of buffer sizes can be adapted to any modeling technique.

In this paper, we study only *consistent acyclic* SDF graphs with initially empty channels (except self-edges).

2.3. Definitions

The *multi-iteration latency* $\mathcal{L}_G(n)$ of the first n iterations of a graph G is equal to the finish time of the last firing of its first n iterations (time is counted from the very first firing).

The *period* $\mathcal{P}_G$ of the execution of a graph G is the average length of an iteration and is formally defined as

$$\mathcal{P}_G = \lim_{n \to \infty} \frac{\mathcal{L}_G(n)}{n} \tag{1}$$

The *throughput* $\mathcal{T}_G$ of a graph G is the number of iterations per unit of time, hence:

$$\mathcal{T}_G = 1/\mathcal{P}_G \tag{2}$$

Eq.(1) and Eq.(2) show the relation between throughput and multi-iteration latency. This is particularly useful in the case of parametric dynamic dataflow models where parameter reconfigurations are frequent. If a given configuration lasts only during m iterations, then $m/\mathcal{L}_G(m)$ gives the achievable throughput for the current configuration.

The *input-output latency* $\ell_G(n)$ of the n^{th} iteration of a graph G is equal to the time between the start time of the first firing and the finish time of the last firing of the n^{th} iteration. The definition given in [Ghamarian et al. 2007] is slightly different but in our context (graphs with initially empty channels) the two definitions are equivalent.

The input-output latency of the complete execution ℓ_G is defined as the maximal latency over all iterations:

$$\ell_G = \max_{n=1..\infty} \ell_G(n) \tag{3}$$

Input-output latency is particularly useful for real-time control systems since it is the maximum delay between sampling data from sensors and sending control commands to the actuators.

For a channel $A \xrightarrow{p \quad q} B$, the i^{th} firing of B (denoted B_i) is enabled if and only if the number of produced tokens is at least $i\,q$. Hence, B has to wait for the j^{th} firing of A (denoted A_j) such that $j\,p \geq iq$. The *data-dependency* between A and B is formalized by the following equation:

$$B_i \geq A_j \quad \text{with} \quad j = \left\lceil \frac{i\,q}{p} \right\rceil \tag{4}$$

The ceiling function in Eq. (4) makes symbolic manipulations difficult. We propose in Section 4 a new characterization that is more intuitive and suitable to reason about buffer sizes and latency.

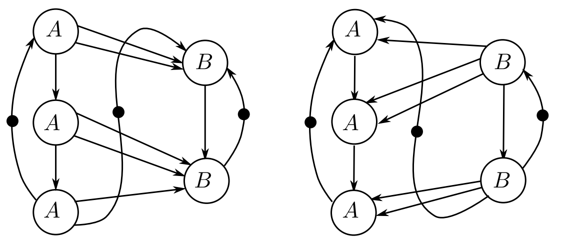
3. THROUGHPUT AND DUALITY

In this section, we first determine the exact maximal throughput for acyclic SDF graphs (**Q1**). Then, we introduce the notion of duality and present a property on dual graphs, used to address the minimum buffer sizes and latency questions.

PROPERTY 3.1 (THROUGHPUT). *The maximal throughput of an acyclic SDF graph $G = (V, E)$ is equal to*

$$\mathcal{T}_G = \frac{1}{\max_{A \in V}\{z_A t_A\}} \tag{5}$$

Hence, the minimal period is $\mathcal{P}_G = \max_{A \in V}\{z_A t_A\}$.



(a) HSDF graph equivalent to $A \xrightarrow[2 \quad 3]{\bullet} B$ (b) HSDF graph equivalent to $B \xrightarrow[3 \quad 2]{\bullet} A$

Fig. 5. SDF-to-HSDF transformation of a graph and its dual.

We say that actor A imposes a higher load than actor B when $z_A t_A > z_B t_B$. The throughput and period of an acyclic graph are therefore defined by the actor that has the highest load, that is actor $\text{argmax}_{A \in V}\{z_A t_A\}$. This implies that this actor never gets idle (*i.e.,* all its firings are consecutive) once the execution enters the steady state.

Definition 3.1. The *dual* of an SDF graph G, denoted G^{-1}, is obtained by reversing all edges of G.

THEOREM 3.2 (DUALITY THEOREM). *Let G be any (possibly cyclic) live SDF graph and G^{-1} be its dual, then $\mathcal{T}_G = \mathcal{T}_{G^{-1}}$ and $\forall i.\ \mathcal{L}_G(i) = \mathcal{L}_{G^{-1}}(i)$.*

We use the transformation of a graph to its dual as well as Theorem 3.2 at several occasions during the analysis of minimal buffer sizes and latency.

4. THE PARAMETRIC GRAPH $A \xrightarrow{p \quad q} B$

This section focuses on the simplest parametric acyclic SDF graph made of a single edge: $G = A \xrightarrow{p \quad q} B$. The graph G is parametrized by the production and consumption rates $p, q \in \mathbb{N}^+$ as well as the execution times $t_A, t_B \in \mathbb{R}^+$. The balance equation $z_A p = z_B q$ entails that the repetition vector of this graph is:

$$[z_A = q/\gcd(p,q), z_B = p/\gcd(p,q)]$$

and, according to Property 3.1, its throughput is:

$$\mathcal{T}_G = \frac{1}{\max(z_A t_A, z_B t_B)} \tag{6}$$

We provide exact symbolic formulas for the minimum buffer size and latency questions. This section shows that the symbolic analysis, even for such simple graphs, is quite involved.

4.1. Enabling patterns

We introduce *enabling patterns*, which characterize the data-dependency between a producer and a consumer. A formal definition can be found in [Bouakaz et al. 2016a]. Compared to Eq. (4), they are better suited to the reasoning about buffer sizes and latency. Intuitively, an enabling pattern between a producer and a consumer describes how firings of the consumer are enabled by the firings of the producer. For example, the enabling pattern of $A \xrightarrow{3 \quad 6} B$ is $A^2 \leadsto B$, meaning that after every two firings of actor A, one firing of B is enabled (B^1 is written B). The enabling pattern of $A \xrightarrow{8 \quad 5} B$ is $A \leadsto B; A \leadsto B^2; A \leadsto B; [A \leadsto B^2]^2$ which is illustrated in Fig. 6. This pattern can also be written as the factorized pattern:

$$\left[A \rightsquigarrow B; [A \rightsquigarrow B^2]^i\right]^{i=1\cdot\cdot 2}$$

This factorized representation is particularly useful when the length and shape of enabling patterns depend on parameters. Depending on the production and consumption rates p and q, there are six possible enabling patterns.

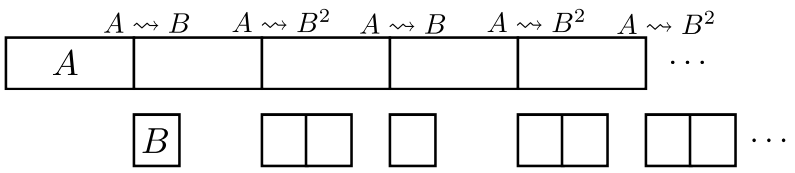


Fig. 6. An ASAP execution of $A \xrightarrow{8 \quad 5} B$ with $t_A = 20$ and $t_B = 7$.

PROPERTY 4.1. *Fig. 7 gathers all possible enabling patterns for the graph* $A \xrightarrow{p \quad q} B$.

Case A. $p \geq q$ Let $p = kq + r$ with $0 \leq r < q$	**Case B.** $p < q$ Let $q = kp + r$ with $0 \leq r < p$
Case A.1. $r = 0$ $$A \rightsquigarrow B^k$$	**Case B.1.** $r = 0$ $$A^k \rightsquigarrow B$$
Case A.2. $q \leq 2r$ $$\left[A \rightsquigarrow B^k; \left[A \rightsquigarrow B^{k+1}\right]^{\alpha_j}\right]^{j=1\cdot\cdot\frac{q-r}{\gcd(p,q)}}$$	**Case B.2.** $p \geq 2r$ $$\left[A^{k+1} \rightsquigarrow B; \left[A^k \rightsquigarrow B\right]^{\gamma_j}\right]^{j=1\cdot\cdot\frac{r}{\gcd(p,q)}}$$
Case A.3. $q > 2r$ $$\left[\left[A \rightsquigarrow B^k\right]^{\beta_j}; A \rightsquigarrow B^{k+1}\right]^{j=1\cdot\cdot\frac{r}{\gcd(p,q)}}$$	**Case B.3.** $p < 2r$ $$\left[\left[A^{k+1} \rightsquigarrow B\right]^{\lambda_j}; A^k \rightsquigarrow B\right]^{j=1\cdot\cdot\frac{p-r}{\gcd(p,q)}}$$
where $\alpha_j = \left\lfloor \frac{jr}{q-r} \right\rfloor - \left\lfloor \frac{(j-1)r}{q-r} \right\rfloor$ and $\beta_j = \left\lceil \frac{jq}{r} \right\rceil - \left\lceil \frac{(j-1)q}{r} \right\rceil - 1.$	where $\gamma_j = \left\lfloor \frac{jp}{r} \right\rfloor - \left\lfloor \frac{(j-1)p}{r} \right\rfloor - 1$ and $\lambda_j = \left\lceil \frac{jr}{p-r} \right\rceil - \left\lceil \frac{(j-1)r}{p-r} \right\rceil.$

Fig. 7. Enabling patterns.

4.2. Minimum buffer size for maximum throughput of $A \xrightarrow{p \quad q} B$

We now use enabling patterns to compute the minimum size of the buffer $A \xrightarrow{p \quad q} B$ (denoted $\theta_{A,B}$) such that the ASAP execution achieves the maximal throughput (given by Eq. (6)) or, equivalently, the minimal period. The buffer size is modeled by adding a backward edge with $\theta_{A,B}$ initial tokens. We distinguish two cases:

• **Case** $z_A t_A \geq z_B t_B$ (*i.e.,* $q t_A \geq p t_B$): Actor A has the highest load and should fire consecutively for maximal throughput. Let δ_j be the minimum number of *initial* tokens in the *backward* edge (representing the buffer size) such that the j^{th} firing of A can occur immediately after the $(j-1)^{th}$ firing of A. By definition of $\theta_{A,B}$, we have $\theta_{A,B} =$

$\max\limits_j \delta_j$. Let x_j denote the number of firings of B that have finished by the start of the j^{th} firing of A. Hence, $\delta_j = jp - x_j q$ and

$$\theta_{A,B} = \max_j (jp - x_j q) \tag{7}$$

The main difficulty when solving symbolically Eq. (7) is to identify an analytic formula for sequence (x_j). Enabling patterns are the key to solve this problem. A trivial case is (A.1) where $p = kq$ and the enabling pattern is $A \rightsquigarrow B^k$. In order for A to fire consecutively, the backward edge should have at least $2p$ tokens. This is because $t_A \geq kt_B$, so the first k firings of B complete before the third firing of A, which still needs $2p$ initial tokens in order to fire again immediately, *i.e.,* $\delta_3 = 2p$. This behavior repeats for all iterations, hence, $\forall i \geq 2.\ \delta_i = 2p$ and the minimum buffer size is $2p$. In general, unlike sequence (x_j), enabling patterns are time-independent. Thus, when considering execution times t_A and t_B, three cases will emerge (cases I, II, and III in Fig. 8); each one has to be solved *w.r.t.* all possible enabling patterns. The three cases should be read as "I else II otherwise III". These cases are described in details in [Bouakaz et al. 2016a]. For instance, case I corresponds to the case where at any given enabling point (*i.e.,* any $\rightsquigarrow$ in the enabling pattern), all newly enabled firings of B complete their execution before the next enabling point.

PROPERTY 4.2. *If $z_A t_A \geq z_B t_B$, then the minimum buffer sizes of $A \xrightarrow{p\quad q} B$ for maximal throughput are given by the symbolic formulas of Fig. 8.*

• **Case** $z_A t_A < z_B t_B$ (*i.e.,* $qt_A < pt_B$): Actor B has the highest load and should fire consecutively for maximal throughput. However, in general not all firings of B are necessarily consecutive since initially, there are no tokens to be consumed. The previous approach can still be followed thanks to the duality theorem. Since the graph G and its dual G^{-1} have the same throughput, we can apply the former reasoning on G^{-1} where B is the producer and has the highest load. Then, Property 4.3 will be used.

PROPERTY 4.3. *If $\theta_{B,A}$ is the minimum buffer size that allows the ASAP execution of G^{-1} to achieve its maximal throughput, then the minimal buffer size $\theta_{A,B}$ for G is such that $\theta_{A,B} = \theta_{B,A}$ (obtained from Property 4.2 by mutual replacement of A by B and of p by q).*

NOTE 4.1. If actors A and B impose the same load (*i.e.,* $z_A t_A = z_B t_B$), then all four cases (III.A.2, III.A.3, III.B.2 and III.B.3) give the same *upper bound*:

$$\theta^u_{A,B} = 2(p + q - \gcd(p, q)) \tag{13}$$

This bound is also tight, in the sense that for all p, q, there exist t_A and t_B such that $\theta_{A,B}$ as given in Fig. 8 is equal to $\theta^u_{A,B}$. This upper bound does not depend on the execution times of the actors. Therefore, it can be used as a safe buffer size if the execution times of actors are unknown.

4.3. Multi-iteration latency of $A \xrightarrow{p\quad q} B$

In this section, we derive analytical formulas for the multi-iteration latency of the first n iterations (*i.e.,* $\mathcal{L}_G(n)$) of graph $A \xrightarrow{p\quad q} B$ (assuming a constant configuration of the parameters). Since our goal is to compute (an approximation of) the maximal achievable throughput, we suppose that buffers are unbounded. There are two cases depending on whether A or B imposes the highest load.

Case I.

Case I.1. A.1 $\vee$ ((A.2 $\vee$ A.3) $\wedge$ $(t_A \geq (k+1)t_B)$)

$$\theta_{A,B} = 2p + q - \gcd(p,q) \tag{8}$$

Case I.2. B.1 $\vee$ ((B.2 $\vee$ B.3) $\wedge$ $(t_B \leq k t_A)$)

$$\theta_{A,B} = p + q - \gcd(p,q) + \left\lceil \frac{t_B}{t_A} \right\rceil p \tag{9}$$

Case II.

Case II.1. (A.2 $\wedge\, r' \geq \frac{\lceil \frac{r}{q-r} \rceil}{\lceil \frac{r}{q-r} \rceil + 1} t_B$) $\vee$ (A.3 $\wedge\, r' \geq \frac{1}{\lfloor \frac{q}{r} \rfloor} t_B$) where $r' = t_A - k t_B$

$$\theta_{A,B} = 2p + q - \gcd(p,q) + \left\lceil \frac{t_B - r'}{r'} \right\rceil r \tag{10}$$

Case II.2. (B.2 $\wedge\, r' \leq \frac{1}{\lceil \frac{p}{r} \rceil} t_A$) $\vee$ (B.3 $\wedge\, r' \leq \frac{\lfloor \frac{r}{p-r} \rfloor}{\lfloor \frac{r}{p-r} \rfloor + 1} t_A$) where $r' = t_B - k t_A$

$$\theta_{A,B} = p + 2q - \gcd(p,q) + \left\lceil \frac{r'}{t_A - r'} \right\rceil (p - r) \tag{11}$$

Case III.

Case III.1. A.2

$$\theta_{A,B} = 2p + q + r - \gcd(p,q) + \max_{j=1}^{n-1}(jr \bmod (q-r)) \tag{12}$$

where n is the smallest positive integer such that $\left\lfloor \frac{nr'}{t_B - r'} \right\rfloor \geq \left\lceil \frac{nr}{q-r} \right\rceil$ and $r' = t_A - k t_B$.

Cases III.(A.3), III.(B.2), III.(B.3) see [Bouakaz et al. 2016a].

Fig. 8. Minimum buffer size $\theta_{A,B}$ when $z_A t_A \geq z_B t_B$.

• **Case** $z_A t_A \geq z_B t_B$, *i.e.,* A imposes a higher load than B. As illustrated in Fig. 9, actor A never gets idle and $\mathcal{P}_G = z_A t_A$. Therefore, we can put

$$\mathcal{L}_G(n) = n\mathcal{P}_G + \Delta_{A,B} \tag{14}$$

such that $\Delta_{A,B}$ is the remaining execution time for actor B after actor A has finished its firings of the n^{th} iteration ($\Delta_{A,B}$ is constant over all iterations). The value of $\Delta_{A,B}$ is given by Property 4.4. Eq.(14) shows that the multi-iteration latency will under-approximate the maximal throughput by only a small amount that decreases with the value of n. Indeed, an over-approximation of the achieved period is

$$\frac{\mathcal{L}_G(n)}{n} = \frac{n\mathcal{P}_G + \Delta_{A,B}}{n} = \mathcal{P}_G + \frac{\Delta_{A,B}}{n} \geq \mathcal{P}_G$$

because Δ_{AB} is non-negative, as can be shown from Property 4.4.

PROPERTY 4.4. *If $z_A t_A \geq z_B t_B$, then the value of $\Delta_{A,B}$ is given by the symbolic formulas of Fig. 10.*

• **Case** $z_A t_A < z_B t_B$, *i.e.,* B imposes a higher load than A. As illustrated in Fig. 11, actor B never gets idle in the steady state. However, in general, not all firings of B are necessarily consecutive since initially there are no tokens on the forward edge $A \to B$.

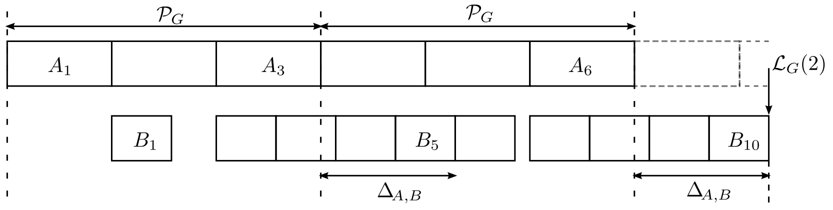


Fig. 9. Multi-iteration latency $\mathcal{L}_G(2)$, case $z_A t_A \geq z_B t_B$ ($p=5$, $q=3$, $t_A=14$, $t_B=8$).

Case I.

$$\Delta_{A,B} = \left\lceil \frac{p}{q} \right\rceil t_B \tag{15}$$

Case II.

Case II.1.

$$\Delta_{A,B} = t_A + \left\lceil \frac{r}{q-r} \right\rceil ((k+1)t_B - t_A) \tag{16}$$

Case II.2.

$$\Delta_{A,B} = t_B + \left\lceil \frac{p-r}{r} \right\rceil (t_B - kt_A) \tag{17}$$

Case III.

Case III.1. Let $r' = t_A - kt_B$ and $n = \frac{q-r}{\gcd(p,q)}$

$$\Delta_{A,B} = t_A + r' + \frac{t_B r - qr'}{\gcd(p,q)} + (t_B - r') \max_{j=0}^{n-1} \left(\frac{jr'}{t_B - r'} - \left\lfloor \frac{jr}{q-r} \right\rfloor \right) \tag{18}$$

Cases III.(A.3), III.(B.2), III.(B.3): see [Bouakaz et al. 2016a].

Fig. 10. Multi-iteration latency: value of $\Delta_{A,B}$.

Note that $\Delta_{A,B}$ is not constant over all iterations and diverges to infinity if the buffer is supposed unbounded. We thus compute $\mathcal{L}_G(n)$ with the duality theorem. We have $\mathcal{L}_G(n) = \mathcal{L}_{G^{-1}}(n)$. Since the producer B in graph G^{-1} imposes the highest load, we have $\mathcal{L}_{G^{-1}}(n) = n\mathcal{P}_{G^{-1}} + \Delta_{B,A}$ where $\Delta_{B,A}$ is computed on the dual graph thanks to Property 4.4.

4.4. Input-output latency of $A \xrightarrow{p \quad q} B$

We now derive analytical formulas for the input-output latency $\ell_G(n)$ of graph $A \xrightarrow{p \quad q} B$. There are two cases depending on which actor imposes the highest load.

• **Case** A imposes the highest load: $\ell_G(n)$ is equal to the finish time of the n^{th} iteration, which is equal to $\mathcal{L}_G(n) = n\mathcal{P}_G + \Delta_{A,B}$ (Eq. (14)), minus the start time of the first firing of A in the n^{th} iteration. This start time is equal to $(n-1)\mathcal{P}_G$ since A is never idle. Hence,

$$\ell_G(n) = \mathcal{L}_G(n) - (n-1)\mathcal{P}_G = \mathcal{P}_G + \Delta_{A,B} \tag{19}$$

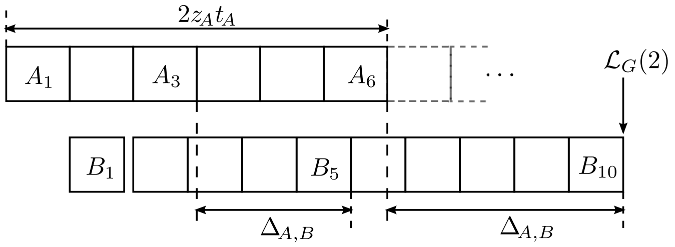


Fig. 11. Multi-iteration latency $\mathcal{L}_G(2)$, case $z_At_A < z_Bt_B$ ($p=5$, $q=3$, $t_A=14$,$t_B=12$).

Hence, $\ell_G = \mathcal{P}_G + \Delta_{A,B} = \mathcal{L}_G(1)$; *i.e.,* the first iteration results in the maximum delay between sampling inputs and sending results.

• **Case** B imposes the highest load: We have $\ell_G(n) = \mathcal{L}_G(n) - (n-1)z_At_A$ if the buffer is unbounded. In this case, the input-output latency diverges with n. However, in practice the buffer is bounded. The buffer size will impact the input-output latency since the firings of A will not be consecutive. As in the previous case, we will assume that the buffer size is larger than $\theta_{A,B}$ to allow the maximal throughput (*i.e.,* B runs consecutively in the steady state). We propose an over-approximation of the maximum input-output latency in Section 5.2, which uses a backward linearization technique.

5. LINEARIZATION OF $A \xrightarrow{p \quad q} B$

In order to use the results of the previous section to obtain approximate analyses of general acyclic dataflow graphs, we propose a technique that *linearizes* the firings of actors. We propose a forward linearization (*i.e.,* linearizing the firings of the consumer) and a backward linearization (*i.e.,* linearizing the firings of the producer).

5.1. Forward linearization of graph $A \xrightarrow{p \quad q} B$

Consider the graph $G = A \xrightarrow{p \quad q} B$, where, as illustrated in Fig. 12, the firings of A are consecutive while those of B are neither consecutive nor uniformly distributed. Let $f_B(i)$ denote the finish time of the i^{th} firing of actor B. In order to derive formulas that can be composed (*e.g.,* to deal with a chain of actors), we transform B into two fictive actors B^u (upper bound) and B^ℓ (lower bound) that both fire consecutively as many times as B does (i.e., $z_{B^u} = z_{B^\ell} = z_B$[3]), and such that

$$\forall i.\ f_{B^\ell}(i) \le f_B(i) \le f_{B^u}(i)$$

Actor B^x (*i.e.,* B^u or B^ℓ) has a starting time $t^0_{B^x}$ and an execution time t_{B^x}, and, since it fires consecutively, $f_{B^x}(i) = it_{B^x} + t^0_{B^x}$.

In the following, we will present tight linearizations, in the sense that $\exists i.\ f_B(i) = f_{B^x}(i)$. For instance, we can see in Fig. 12 that the 5^{th} firings of B and B^s finish at the same time.

Intuitively, thanks to these linearizations, a chain $A \xrightarrow{p \quad q} B \xrightarrow{p' \quad q'} C$ such that $z_At_A > z_Bt_B > z_Ct_C$ can be treated by first scheduling the subgraph $A \xrightarrow{p \quad q} B$, then linearizing the firings of B (which are not consecutive since $z_At_A > z_Bt_B$) to obtain B^x, then scheduling the subgraph $B^x \xrightarrow{p' \quad q'} C$, and finally combining the two schedules. Thanks to this approach, we can compute a safe upper bound of the minimum buffer sizes for

[3]Actually, this is valid if we compute over a single iteration. Regarding the multi-iteration latency, we linearize over a fixed number n of iterations.

a chain $A \to B \to C$, instead of trying to generalize the formulas of Fig. 8 for the two edges, which is too complex.

5.1.1. Upper bound linearization. We present two linearization methods, *Push* and *Stretch*, illustrated in Fig. 12. *Push* is applied to n successive iterations: it considers the actor B^p which is obtained by pushing *all* the firings of B in n iterations to the right end to get rid of all the gaps. The execution time remains the same (*i.e.,* $t_{B^p} = t_B$) but the start time of the first of the consecutive firings varies with the number of iterations n considered. It is equal to the multi-iteration latency $\mathcal{L}_G(n)$ minus the execution time of nz_B firings: $t^0_{B^p} = \mathcal{L}_G(n) - nz_Bt_B$.

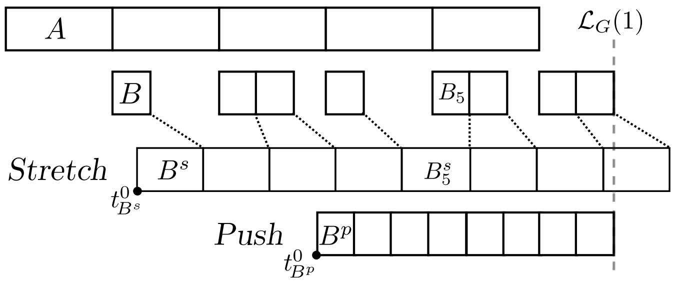


Fig. 12. Upper bound linearization ($p = 8$, $q = 5$, $t_A = 20$, $t_B = 7$).

The second method, *Stretch*, builds the actor B^s by increasing the execution time of B in order to fill the gaps over an infinite execution. We distinguish two cases:

• **Case** $z_At_A \geq z_Bt_B$: This case is illustrated in Fig. 12. We fix the load of B^s to be the same as A, that is $t_{B^s} = \frac{qt_A}{p}$. Then $t^0_{B^s}$, the starting time of B^s, is computed as follows: for all i, we have $f_{B^s}(i) = it_{B^s} + t^0_{B^s}$; since we also want $f_{B^s}(i) \geq f_B(i)$ to hold for all i, it follows that $\forall i.\ t^0_{B^s} \geq f_B(i) - it_{B^s}$; hence $t^0_{B^s} = \max_i(f_B(i) - it_{B^s})$. As in the case of the minimum buffer size problem, we have to consider the three cases (I), (II) and (III) and all six enabling patterns. It can be shown (see [Bouakaz et al. 2016a]) that $t^0_{B^s} = t_A + t_B - \frac{\gcd(p,q)}{p}t_A$. We conclude that:

$$\forall i.\ f_{B^s}(i) = it_{B^s} + t^0_{B^s} = \frac{qt_A}{p}i + \left(t_A + t_B - \frac{\gcd(p,q)}{p}t_A\right) \tag{20}$$

Method *Stretch* may move the starting of some firings earlier (*e.g.,* the 2^{th} firing of B^s in Fig. 12), but it always postpones their endings. This is why it is a safe over-approximation for computing the multi-iteration latency.

• **Case** $z_At_A < z_Bt_B$: In this case, methods *Push* and *Stretch* are identical. The firings of B are consecutive in the steady state. Therefore, we can take $t_{B^s} = t_{B^p} = t_B$ and, using the duality theorem, we have $\mathcal{L}_G(n) = \mathcal{L}_{G^{-1}}(n) = nz_Bt_B + \Delta_{B,A}$ and we can take $t^0_{B^s} = \Delta_{B,A}$, where $\Delta_{B,A}$ is computed on the dual graph (Property 4.4).

5.1.2. Lower bound linearization. Here we use the *Stretch* method: the execution time of B is increased in order to fill the gaps over an infinite execution, yielding a new actor B^ℓ such that $\forall i.\ f_{B^\ell}(i) = it_{B^\ell} + t^0_{B^\ell} \leq f_B(i)$, *i.e.,* the finishing times of B^ℓ are moved earlier. This implies that the the starting times of B^ℓ may also be moved earlier. Again, we distinguish two cases:

• **Case** $z_A t_A \geq z_B t_B$: We have $t_{B^\ell} = \frac{q t_A}{p}$ and $t^0_{B^\ell} = \min_i (f_B(i) - i t_{B^\ell})$. Fig. 13 shows the symbolic formulas for $t^0_{B^\ell}$ (the proof is in [Bouakaz et al. 2016a]).

Case A. $p \geq q$

Let $p = kq + r$ with $0 \leq r < q$

$$t^0_{B^\ell} = k t_B + \min \left\{ t_B, \frac{r t_A}{p} \right\} \tag{21}$$

Case B. $p < q$

Let $q = kp + r$ with $0 \leq r < p$ and $\sigma = p t_B - q t_A$

Case I.

$$t^0_{B^\ell} = t_B \tag{22}$$

Cases II + III.

$$t^0_{B^\ell} = t_B + \frac{p - r}{p} t_A + \left\lfloor \frac{p}{r} \right\rfloor \frac{\sigma}{p} + \min \left\{ \frac{-\sigma}{p}, \frac{r - (p \bmod r)}{p} t_A \right\} \tag{23}$$

Fig. 13. Lower bound linearization in case $z_A t_A \geq z_B t_B$.

• **Case** $z_A t_A < z_B t_B$: This case is equivalent to a push to the left, that is, deriving a fictive actor B^ℓ such that $t_{B^\ell} = t_B$, B^ℓ_0 has the same starting time as B_0, and all firings of B^ℓ are consecutive. Hence, $t^0_{B^\ell}$ is equal to t^0_B.

5.2. Backward linearization of graph $A \xrightarrow{p \quad q} B$

In order to compute the input-output latency of chains of actors, we also propose a backward lower bound linearization of the producer. Here the goal is to make the firings of A consecutive. If A imposes a higher load than B, then the backward linearization is trivial since the firings of A are already consecutive, assuming that the buffer size allows the throughput to be maximal.

Suppose now that B imposes a higher load than A and that the channel is large enough to allow B to run consecutively in the steady state (*i.e.,* the ASAP execution achieves its maximal throughput). Hence, a safe buffer size will be $\theta_{A,B}$ (Property 4.2).

Let $s_X(i)$ denote the start time in the ASAP schedule of the i^{th} firing of actor X. We want to transform actor A into a fictive actor A^ℓ with consecutive firings such that $\forall i.\ s_{A^\ell}(i) \leq s_A(i)$; *i.e.,* start times are moved backward. This constraint is sufficient in this case to guarantee an over-approximation of the input-output latency. However, if channel $A \to B$ is a part of a chain (say $Z \to A \to B$), then we also need to ensure that the finish times of A^ℓ are not postponed; otherwise, this may impact the schedule of graph $Z \to A^\ell$ by delaying the firings of Z (due to the buffer size constraint) and hence *under-approximating* the total input-output latency. Therefore, the required linearization constraint is rather:

$$\forall i.\ f_{A^\ell}(i) \leq f_A(i) \tag{24}$$

PROPERTY 5.1. *If $z_A t_A < z_B t_B$, then a valid backward lower bound linearization of A is given by*

$$f_{A^\ell}(i) = i t_{A^\ell} + (t^0_{A^\ell} + s_B(j_0 + 1) - i_0 t_{A^\ell}) \tag{25}$$

where (i_0, j_0) *is a solution of the equation* $i_0 p - j_0 q = d$ *(*d *being the smallest buffer size guaranteeing the maximal throughput), and* t_{A^ℓ} *and* $t^0_{A^\ell}$ *are the results of the forward lower bound linearization of* A *in the dual graph* G^{-1}.

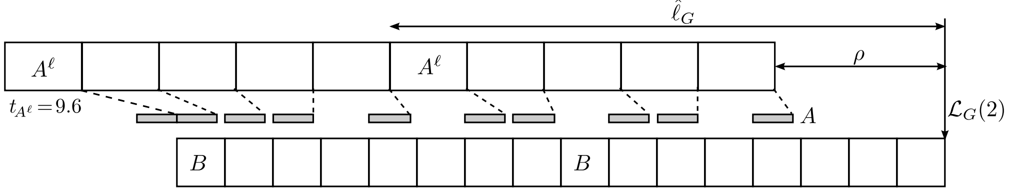


Fig. 14. Backward lower bound linearization (p=8, q=5, t_A=5, t_B=6).

We now show how to use Property 5.1 to compute an upper bound of the input-output latency of graph $A \xrightarrow{p \quad q} B$ when B imposes a higher load than A. Fig. 14 illustrates the ASAP execution of the graph $G = A \xrightarrow{8 \quad 5} B$ such that $t_A = 5$, $t_B = 6$ and the buffer size d is equal to 22. Actor A^ℓ represents the backward lower bound linearization of A.

Let ρ denote the length of the interval between the finish time of the firings of B after n iterations, *i.e.,* $\mathcal{L}_G(n)$, and the finish time of the firings of A^ℓ also after n iterations, *i.e.,* $f_{A^\ell}(nz_A)$. So, $\rho = \mathcal{L}_G(n) - f_{A^\ell}(nz_A)$, which is constant for all n. An over-approximation of the maximum input-output latency is therefore:

$$\hat{\ell}_G = \mathcal{P}_G + \rho \quad \text{with (see electronic appendix)} \quad \rho = \frac{d}{q} t_B - t^0_{A^\ell} \tag{26}$$

The following sections rest upon these linearization techniques. The forward upper bound linearization techniques, *Push* and *Stretch* are used to compute an over-approximation of the multi-iteration latency; *Stretch* is also used to over-approximate buffer sizes. Forward lower bound linearization is used as an intermediate technique to compute the backward lower bound linearization which is applied to compute an over-approximation of the input-output latency.

6. BUFFER SIZING FOR ACYCLIC GRAPHS

Exact symbolic buffer sizing for a single edge graph is already so complex that it seems to be out of reach for arbitrary (even acyclic) graphs. This section shows how to use the previous results to obtain *approximate* analyses for the minimum buffer sizes of general acyclic dataflow graphs in order to reach the maximal throughput. To achieve this, we make use of the forward linearization techniques.

We first present formulas to compute safe upper bounds for general DAGs, then we present a heuristic that improves this bound for chains, trees (a DAG with only forks), and in-trees (a DAG with only joins). These kinds of graphs, especially chains, are common in streaming applications. Finally, we present the exact *numerical* analysis that is used later to evaluate our approximate analyses.

6.1. Safe upper bounds

We first present a negative result. Let G be an acyclic graph and let the size of each channel $A \xrightarrow{p \quad q} B$ be equal to $\theta_{A,B}$ as defined in Section 4.2. These buffer sizes do not always permit maximal throughput (they do however allow maximal throughput in some specific cases described in Section 6.2). Indeed, the computation of $\theta_{A,B}$ assumes that A runs consecutively. This is not always the case if A is constrained by some input data-dependencies.

PROPERTY 6.1. *Let G be a graph without any undirected cycle, if the buffer of every channel $A \xrightarrow{p \quad q} B$ in G is at least $\theta^u_{A,B} = 2(p + q - \gcd(p, q))$, then the ASAP execution of the graph achieves the maximal throughput*[4].

NOTE 6.1. Since the minimum buffer sizes below which the graph is definitely not live are equal to $p + q - \gcd(p, q)$ [Battacharyya et al. 1996], Property 6.1 provides a first solution that is less than twice the exact one. For parametric dataflow models, the upper bound $\theta^u_{A,B}$ can actually be reached for some configurations. However, if the system supports dynamic reallocation of memory, it is still useful to evaluate the minimal buffer sizes in order to adjust the buffers sizes after each configuration change.

Unfortunately, Property 6.1 does not hold for general acyclic graphs that contain undirected cycles. A counterexample is the graph $G_c = \{A \xrightarrow{4 \quad 3} B \xrightarrow{3 \quad 8} D,\ A \xrightarrow{1 \quad 3} C \xrightarrow{3 \quad 2} D\}$ with $t_A = 4$, $t_B = 3$, $t_C = 12$ and $t_D = 8$. The repetition vector is $\vec{z} = [6, 8, 2, 3]$ and all actors impose the same load (*i.e.,* $\forall X.\ z_X t_X = 24$). The ASAP execution when all buffer sizes are equal to their upper bound $2(p + q - \gcd(p, q))$ is shown in Fig. 15. Actor A does not fire consecutively so the throughput is not maximal. The reason is that the chain $A \to C \to D$ imposes an earliest start time for D that is greater than the earliest start time imposed by the chain $A \to B \to D$. More precisely, the first firing of actor D is delayed by actor C, which delays the 7^{th} firing of B, which in turn delays the 8^{th} firing of A. Let $f^u_{D,1}$ (resp. $f^u_{D,2}$) denote the forward linear upper bound on the finish times of actor D following the first (resp. second) chain. We have $f^u_{D,1}(i) = 8i + 16$ and $f^u_{D,2}(i) = 8i + 28$, hence $f^u_{D,2}(i) > f^u_{D,1}(i)$. In order to prevent the second chain $A \to C \to D$ from impacting the schedule of the first chain $A \to B \to D$, we must increase the size of buffer $B \to D$ so that B can fire without being blocked during $f^u_{D,2}(i) - f^u_{D,1}(i) = 28 - 16$ time units. Since B produces 3 tokens per firing and $t_B = 3$, the size of the $B \to D$ buffer must be increased by $\lceil \frac{28-16}{3} \rceil \times 3 = 12$.

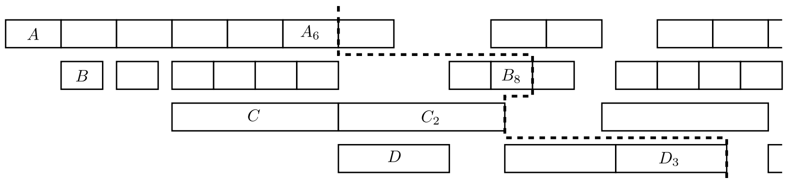


Fig. 15. ASAP execution of G_c; $\vec{z} = [6, 8, 2, 3]$.

In general, if the actors of a chain $A_1 \to A_2 \to \ldots \to A_n$ do not have the same load, we consider, as an upper-bound case, the chain where all actors' loads are increased to be equal to the maximum load of the original chain, obtained by applying the Stretch method to all actors, *i.e.,* the upper-bound forward linearization. Then, Eq. (20) can be applied transitively on edges of this chain to compute a forward upper bound linearization of A_n.

PROPERTY 6.2. *Let two different chains from A_1 to A_n such that $f_{A^u_n,1}(i) = t_{A_n} i + s_1$, $f_{A^u_n,2}(i) = t_{A_n} i + s_2$ with $s_1 < s_2$, and such that the size of each buffer $A_i \xrightarrow{p_i \quad q_i} A_{i+1}$ is equal to $\theta^u_{A_i,A_{i+1}} = 2(p_i + q_i - \gcd(p_i, q_i))$. In order to prevent the second chain from*

[4]This property generalizes the double-buffering technique used for HSDF graphs.

disturbing the schedule of the first one, it suffices to increase the size of the last channel $A_{n-1} \xrightarrow{p_{n-1} \quad q_{n-1}} A_n$ of the first chain by ζ:

$$\zeta = \left\lceil \frac{s_2 - s_1}{t_{A_{n-1}}} \right\rceil p_{n-1} \tag{27}$$

Actually, ζ can be computed in different ways (see [Bouakaz et al. 2016a]). This approach can be applied to any acyclic graph with undirected cycles as shown in Algorithm 1. Such an approach is safe but not always needed (*e.g.,* when the predecessors of a node do not have common ancestors).

ALGORITHM 1: Safe upper bounds for graphs with undirected cycles

```
Input: SDF graph $G$ with undirected cycles, all actors impose the same load
Output: Safe buffer sizes
Data: $L$ = list of actors of $G$ in topological order
while $L \neq \emptyset$ do
    $B = dequeue(L)$;
    $pred(B) = incoming\text{-}edges(B)$;
    if $pred(B) = \emptyset$ then $s_B = 0$ ;
    else
        for each edge $A \xrightarrow{p \quad q} B \in pred(B)$ do $s_{B,A} = s_A + \frac{t_B}{q}(p + q - \gcd(p,q))$ ;
        $s_B = \max_{A \xrightarrow{p \quad q} B \in pred(B)} (s_{B,A})$;
        for each edge $A \xrightarrow{p \quad q} B \in pred(B)$ do $size(A \xrightarrow{p \quad q} B) = \theta^u_{A,B} + \left\lceil \frac{s_B - s_{B,A}}{t_{A_{n-1}}} \right\rceil p$ ;
```

6.2. Improving the upper bounds

In this section, we improve the minimum buffer sizes for chains, trees, and in-trees, starting with chains. We say that a chain is *monotone* iff either each actor imposes a higher load than its successor, or each actor imposes a lower load than its successor.

Definition 6.1. The chain $A_1 \to \cdots \to A_n$ is *monotone* if and only if $(\forall i.\ z_{A_i} t_{A_i} \geq z_{A_{i+1}} t_{A_{i+1}}) \vee (\forall i.\ z_{A_i} t_{A_i} \leq z_{A_{i+1}} t_{A_{i+1}})$

PROPERTY 6.3. *A monotone descending chain $A_1 \to \cdots \to A_n$ where the size of each buffer $A_i \to A_{i+1}$ is at least $\theta_{A_i,A_{i+1}}$ (Property 4.2) achieves its maximal throughput.*

First, note that Property 6.3 also holds for monotone ascending chains by duality. Note also that Property 6.3 is only a *sufficient* condition, because $\theta_{A_i,A_{i+1}}$ allows actor A_i to fire consecutively. However, it is not a necessary condition to achieve the maximal throughput.

Property 6.3 also holds for non monotone chains made of an ascending sub-chain followed by a descending one. We say that those chains are of the form $\sqcap$. The computed buffer sizes on both sub-chains allow the actor(s) at the "top" of the $\sqcap$ to achieve its maximal throughput (in full generality, there can be several such actors with identical load).

Unfortunately, Property 6.3 does not hold for an arbitrary chain (*i.e.,* neither ascending, descending nor of the form $\sqcap$). Our solution is to put such an arbitrary chain under the form $\sqcap$: we increase the execution times of some actors (without exceeding the maximum load $\mathcal{P}_G$), then we compute the buffer sizes as in Property 6.3, and finally we restore the original execution times. Fig. 16 illustrates this solution.

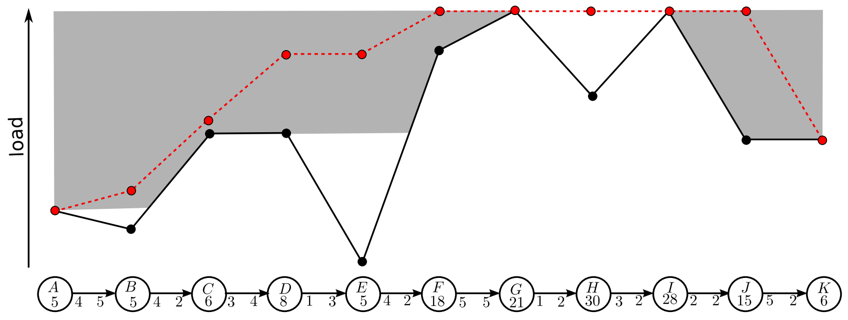


Fig. 16. Transformation of a chain to a $\sqcap$ form.

Any chain on the form $\sqcap$ obtained by increasing the load of the actors of the original chain is a valid solution. For example, the chain $A \rightarrow B \rightarrow \ldots \rightarrow K$ of Fig. 16 can be transformed into the red chain which is of the form $\sqcap$. Actually, any chain of this form inside the gray area is a valid solution. An interesting problem is to find one that minimizes the sum of the buffer sizes. We have proposed three possible solutions.

• *MAX:* All loads are raised up to the maximal load (*i.e.,* the top boundary of the gray area). This case is identical to Property 6.1, *i.e.,* the size of each channel $A \xrightarrow{p \quad q} B$ will be equal to $\theta^{u}_{A,B} = 2(p + q - \gcd(p, q))$.

• *MIN:* All loads are raised up to the bottom boundary of the gray area. In Fig. 16, the load of B will be increased to that of A, the load of E to that of D, and the load of H to that of G.

• *OPT*: The third solution is an optimization heuristic based on two observations: (i) The minimum difference between the loads of A and B that allows to optimize the size of $A \xrightarrow{p \quad q} B$ can be deduced from Fig. 8. (ii) The maximum gain on a channel size that comes from changing the loads depends on the production and the consumption rates of this channel. This expected size gain is used to prioritize the treatment of channels.

More details on the *OPT* heuristic and an experimental comparison of the three algorithms can be found in [Bouakaz et al. 2016a].

The case of trees is solved in the same way. If the tree does not contain any sub-trees (*i.e.,* it consists of a set of chains originating from the same root node), then the load of the root node is first increased to be equal to the maximum of all loads in the tree and then the previous method can be applied on every chain composing the tree. This is correct because the computed buffer sizes will allow the root actor to run consecutively, thus guaranteeing that the execution reaches the maximum throughput. If the tree contains sub-trees, the same process is first applied recursively on sub-trees, and then we proceed by replacing each sub-tree by its root node. In-trees are dealt with by using the duality theorem.

7. LATENCY COMPUTATION FOR ACYCLIC GRAPHS

Like Section 6, this section shows how we can use the results for a single edge graph $A \xrightarrow{p \quad q} B$ to obtain *approximate* analyses for the latency of general acyclic dataflow graphs. To achieve this, we make use of the linearization techniques (Section 5.1).

7.1. Multi-iteration latency of acyclic graphs

In this section, we compute an *upper bound* of the multi-iteration latency of the first n iterations, denoted $\mathcal{L}_G(n)$. Since we are interested in the latency, all the buffer sizes in

G are assumed to be infinite. A similar approach can be used to compute a lower bound. However, upper bounds are more useful in practice since they ensure important safety properties (*e.g.,* a deadline, a minimal quality of service, *etc.*). Here, the computed upper bound is used to under-approximate the maximal achievable throughput.

The principle is first to extract all the maximal chains from G, then to use an upper bound forward linearization to compute the latency of each such chain, and to find the best linearization that minimizes the over-approximation of the latency.

Any acyclic SDF graph G can be represented as a set of *maximal chains* $\mathcal{G}(G)$, that is, chains from a source actor to a sink actor. By considering each chain $g \in \mathcal{G}(G)$ as an SDF graph[5], we have the following property:

PROPERTY 7.1. *For any acyclic SDF graph G, $\forall i.\ \mathcal{L}_G(i) = \max_{g\in\mathcal{G}(G)}\{\mathcal{L}_g(i)\}$*

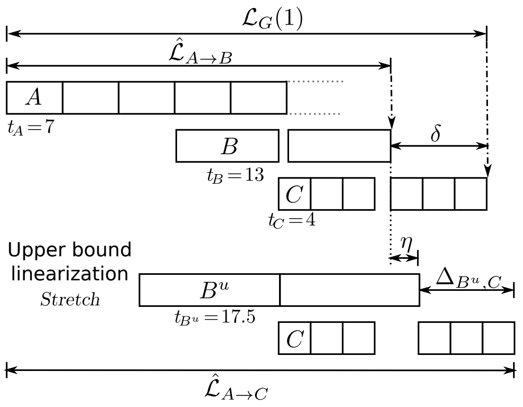


Fig. 17. Computation of $\hat{\mathcal{L}}_{A\to C}$ of a chain $A \xrightarrow{2\ \ 5} B \xrightarrow{3\ \ 1} C$.

Using Property 7.1, the problem reduces to computing the multi-iteration latency of each chain in G. For each chain $A \xrightarrow{p_1\ \ q_1} B \xrightarrow{p_2\ \ q_2} C \to \cdots \to Z$, we compute an upper bound of its multi-iteration latency for n iterations, denoted by $\hat{\mathcal{L}}_{A\to Z}$ (we omit n for the sake of conciseness). We can compute exactly $\mathcal{L}_{A\to B}$ as described in Section 4.3. However, since the technique assumes that the producer can run consecutively, in general it cannot be applied between B and C. We compute an upper bound linearization of the firings of B such that they are consecutive and $\forall j \leq nz_B.\ f_{B^u}(j) \geq f_B(j)$.

As illustrated in Fig. 17, the exact multi-iteration latency of chain $A \to B \to C$ is $\mathcal{L}_G(n) = \hat{\mathcal{L}}_{A\to B} + \delta$ where δ is the remaining execution time of C after the end of B.

Let $\eta = f_{B^u}(nz_B) - \hat{\mathcal{L}}_{A\to B}$ (*i.e.,* the approximation introduced by the upper bound linearization, see Fig. 17), then $\delta \leq \Delta_{B^u,C} + \eta$. Indeed, $\Delta_{B^u,C}$ (computed by Equations of Fig. 10) gives the remaining time for C after the end of B^u (recall that firings of B^u are consecutive and hence the method described in Section 4.3 can be used).

The critical part of this method is to find the best upper bound linearization that minimizes the difference between the exact value δ and the approximate one $(\Delta_{B^u,C} + \eta)$. We propose two upper bound linearization methods, *Push* and *Stretch* (Section 5.1.1). It can be shown that the two methods are incomparable even if we distinguish the cases when B imposes a higher load than C and vice-versa. In both cases, there are graphs for which either *Push* or *Stretch* is better. Since the two methods are not costly to try, we apply both and take the minimum.

[5]When computing the latency of g, the repetition count of each actor X in g is made equal to $\vec{z}_G(X)$.

In the case of *Push*, we have

$$\hat{\mathcal{L}}_{A\to C} \leq \hat{\mathcal{L}}_{A\to B} + \eta + \Delta_{B^p,C} = \hat{\mathcal{L}}_{A\to B} + 0 + \Delta_{B,C}$$

In the case of *Stretch* (Fig. 17), we have

$$\begin{aligned}\hat{\mathcal{L}}_{A\to C} &\leq \hat{\mathcal{L}}_{A\to B} + \eta + \Delta_{B^s,C} \\ &= \hat{\mathcal{L}}_{A\to B} + (f_{B^s}(nz_B) - \hat{\mathcal{L}}_{A\to B}) + \Delta_{B^s,C} = f_{B^s}(nz_B) + \Delta_{B^s,C}\end{aligned}$$

Therefore, we have

$$\hat{\mathcal{L}}_{A\to C} \leq \min\{\hat{\mathcal{L}}_{A\to B} + \Delta_{B,C}, f_{B^s}(nz_B) + \Delta_{B^s,C}\} \tag{28}$$

The same process can be repeated to treat an arbitrary long chain. For instance, to compute the latency of the sub-chain $A \to B \to C \to D$, we have $\hat{\mathcal{L}}_{A\to D} = \min\{\hat{\mathcal{L}}_{A\to C} + \Delta_{C,D}, f_{C^s}(nz_C) + \Delta_{C^s,D}\}$ such that $\hat{\mathcal{L}}_{A\to C}$ is the latency computed in the previous step (*i.e.,* for sub-chain $A \to B \to C$), and C^s is the linearization of C using the method *Stretch* applied transitively on actors of the sub-chain $A \to B \to C$.

NOTE 7.1. Instead of analyzing separately all the chains of a DAG, which may have an exponential complexity, it is more efficient to use the compositionality of our approach to prevent some recomputations. For instance, if we have two chains $A \to B \to D \to E$ and $A \to C \to D \to E$ (*i.e.,* actor D is a join), then we merge the information that comes from both paths: $\hat{\mathcal{L}}_{A\to D}$ is taken as the maximum of $\hat{\mathcal{L}}_{A\to B\to D}$ and $\hat{\mathcal{L}}_{A\to C\to D}$. An algorithm inspired from longest path algorithms and using compositionality should achieve polynomial time complexity.

NOTE 7.2. According to the duality theorem, the multi-iteration latencies of a chain $A \to \cdots \to Z$ and of its dual are equal. However, our method may give different approximate values, *i.e.,* $\hat{\mathcal{L}}_{A\to Z} \neq \hat{\mathcal{L}}_{Z\to A}$. Therefore, for a given chain, we analyze both the chain and its dual and return $\min\{\hat{\mathcal{L}}_{A\to Z}, \hat{\mathcal{L}}_{Z\to A}\}$. Again, since both computations have a linear complexity, this is not costly.

NOTE 7.3. Knowing whether the minimum buffer sizes to guarantee the maximal throughput will also allow the graph to achieve its minimal multi-iteration latency is an open problem.

7.2. Input-output latency of chains

We now compute the maximum input-output latency ℓ_G (or an upper bound $\hat{\ell}_G$) of a chain G. The input-output latency of the n^{th} iteration, $\ell_G(n)$ is equal to the difference between the multi-iteration latency $\mathcal{L}_G(n)$ and the start time of the first firing of the source actor in the n^{th} iteration.

If the source actor A imposes the highest load among all actors of the graph or if all the channels are unbounded, then the source actor never gets idle and achieves the maximal throughput. Hence, we can put

$$\ell_G(n) = \mathcal{L}_G(n) - (n-1)z_A t_A \tag{29}$$

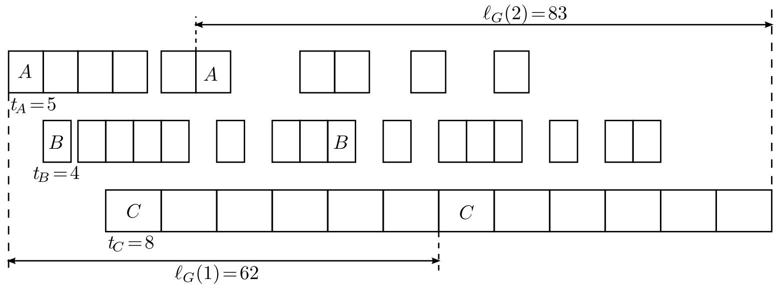
If the source actor does not impose the highest load, then $\ell_G(n)$ as given by Eq. (29) is unbounded unless the channels are bounded. Therefore, as in the case of the graph $A \xrightarrow{p \quad q} B$, we need to consider the buffer sizes when computing the input-output latency.

Consider for instance the chain $A \xrightarrow{8 \quad 5} B \xrightarrow{3 \quad 4} C$ with $t_A = 5, t_B = 4$ and $t_C = 8$. The size of channel $A \to B$ is 24 and the size of channel $B \to C$ is 12. Actor C imposes the highest load. Fig. 18(a) shows the ASAP schedule for two iterations. We have $\hat{\ell}_G =$

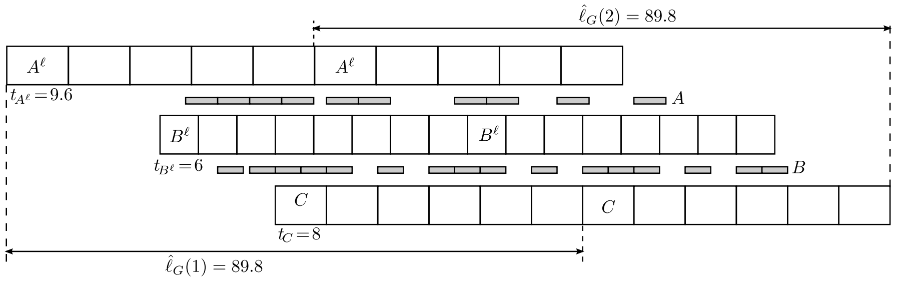
$\ell_G(2) = 83$. Note that the sink actor C runs consecutively but not the source actor A, which prevents us from using Eq. (29) to compute the input-output latency.

As illustrated in Fig. 18(b), our solution consists in using a lower bound backward linearization (Section 5.2). Starting by the end of the chain (channel $B \to C$), actor B is first transformed into a fictive actor B^ℓ that runs consecutively such that $\forall i. f_{B^\ell}(i) \leq f_B(i)$. This constraint also implies that the start times of the firings of B^ℓ are advanced compared to the start times of B (because $t_{B^\ell} \geq t_B$), which leads to an over-approximation of the input-output latency.

Since B^ℓ runs consecutively and imposes a higher load than A, the same process can be repeated to linearize A backward. It follows that the start times of the firings of A^ℓ in the final schedule are an under-approximation of the actual start times. The computation of the input-output latency is now straightforward (using Eq. (29)) since the input actor A^ℓ runs consecutively. So, we have $\hat{\ell}_G = 89.8$, which is only an 8.2% over-approximation compared to the actual input-output latency $\ell_G = 83$.



(a) Actual schedule



(b) Linearized schedule

Fig. 18. Input-output latency computation for the chain $A \xrightarrow{8\ \ 5} B \xrightarrow{3\ \ 4} C$.

The previous approach can be applied to any arbitrary chain where the sink actor imposes the highest load. If the actor with highest load[6], denoted H, is in the middle of a chain $A \to \ldots \to H \to \ldots \to Z$, then the lower bound linearization of A (*i.e.,* A^ℓ) is computed using the above described backward linearization starting from actor H, while the upper bound linearization of Z (*i.e.,* Z^u) is computed using the forward linearization (Section 7.1) starting from actor H. An over-approximation of the input-output latency can be then computed between A^ℓ and Z^u.

[6]If there are many, then we take any of them.

8. EXPERIMENTS

Detailed experiments can be found in [Bouakaz et al. 2016a]. This section only outlines the main results.

8.1. Buffer sizing

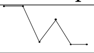
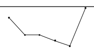
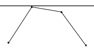
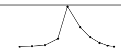
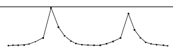
We have compared the results of the three algorithms (*MAX*, *MIN* and *OPT*) presented in Section 6.2 with each other and with the exact minimum buffer sizes using many randomly generated SDF graphs and some real benchmarks.

We have compared the results of algorithm *MIN* with those of *MAX* (*i.e.,* safe upper bounds $2(p+q-\gcd(p,q)))$ on millions of randomly generated chains. The experiments show that, on average, algorithm *MIN* reduces the total buffer sizes by 8% compared to the upper bounds (algorithm *MAX*). Similarly, we have compared algorithm *OPT* with both *MAX* and *MIN* algorithms on randomly generated chains. The experiments show that, on average, *OPT* improves over *MAX* by almost 11.1% and improves over *MIN* by almost 3.5%. This is a significant improvement knowing that transferred tokens in streaming applications could be blocks of video frames. However, if one is looking for time efficiency (which could be important for an online computation for instance), then *MIN* is less expensive than *OPT*.

We have also compared the results of algorithm *OPT* with the exact minimum buffer sizes computed by the enumeration algorithm (which we denote by *EXACT*). Due to the exponential complexity of the minimum buffer sizes problem, we evaluate our approach on only ten thousand randomly. In average, algorithm *OPT* over-approximates the exact solution by 25%. Furthermore, the experiment shows that *MAX* over-approximates *EXACT* by 55% in average.

Finally, we evaluated the heuristics using five real applications: the H.263 decoder, the data modem and sample rate converter from the SDF^3 benchmarks [Stuijk et al. 2006b], the fast Fourier transformer (FFT), and the time delay equalizer (TDE) from the StreamIt benchmarks [Thies and Amarasinghe 2010]. All these graphs have a chain structure. Table I shows some characteristics of these applications together with the obtained results. Our approach improves better the upper bounds in case of chains with a $\prod$ form (H.263 decoder and FFT). It comes close to the upper bound for the sample rate converter since the two actors with the highest loads are the right and left ends of the chain; increasing the loads of the other actors to get a monotone chain results in a size of almost $2(p+q-\gcd(p,q))$ for every channel.

Table I. Experimental results for real benchmarks.

graph	# actors	$\sum_A z_A$	load shape	*MAX*	*EXACT*	*OPT*
modem	6	37		32	20	30
sample rate conv.	6	612		60	34	57
H.263 decoder	4	1190		2378	1257	1257
FFT	11	94		992	504	808
TDE	27	2867		7328	3680	5272

8.2. Latency computation

We have evaluated our approach for computing the multi-iteration latency using millions of randomly generated chains. The experiments show that the average over-approximation is negligible when the number of firings per iteration of the graph is small. Indeed, if there are many harmonious rates (recall that, when p divides q or q divides p, the computed latency for $A \xrightarrow{p\quad q} B$ is exact), then the computed latency remains close to the exact value. Then, the average over-approximation increases to reach its peak (approximately 2.5%) at around fifty firings per iteration. This is because the exact values of latency at these points are small and hence the over-approximation is more noticeable. Then, the average over-approximation decreases and converges to zero for graphs with large latencies. These observations were confirmed by many other experiments (*e.g.,* with longer chains, larger rates, *etc.*) not reported in this paper. The experiment also shows that using only the *Stretch* linearization method is better (in average) than using only the *Push* method. It also shows that using both methods on all channels of the chain (line *Push+Stretch*) is better than taking the minimum of their separate results (line *min*{*Push, Stretch*}). The results are further improved by using the duality theorem (line *Push+Stretch+Dual*) as explained in note 7.2.

Table II presents the obtained results for the real benchmarks. It shows that our approach gives exact results for most of these benchmarks. Production and consumption rates of channels of these graphs are quite harmonious (p divides q or q divides p), for which our approach performs very well, as noticed in the previous experiment.

Table II. Multi-iteration latency computation for real benchmarks.

graph	$\mathcal{P}_G$	$\mathcal{L}_G(1)$	$\hat{\mathcal{L}}_G(1)/\mathcal{L}_G(1)$	$\hat{\mathcal{L}}_G(2)/\mathcal{L}_G(2)$
(a) modem	32	62	1	1
(b) sample con.	960	1000	1.022	1.011
(c) H.263 dec.	332046	369508	1	1
(d) FFT	78844	94229	1	1
(e) TDE	17740800	19314069	1	1

Finally, we evaluate our approach for computing the input-output latency using 10^5 randomly generated chains. The experiment shows that our analysis over-approximates the exact computation, on average, by at most 13%. The over-approximation is less noticeable for graphs with large input-output latencies.

9. RELATED WORK

Few symbolic results about SDF graphs can be found in the literature. In this section, we present the most relevant ones. Consistency can easily be checked analytically. The repetition vector can be computed symbolically as is it done in most dynamic parametric SDF models (*e.g.,* [Bebelis et al. 2013; Fradet et al. 2012]).

There is no exact analytic solution to check the liveness of a graph with buffers with fixed bounds. In [Bebelis et al. 2013] and [Bempelis 2015], the authors apply Eq. (4) transitively (which leads to nested ceilings) on edges of each cycle in the graph. Then, the obtained equations are linearized by over-approximating the ceiling function (*i.e.,* $\lceil x \rceil < x + 1$). However, this is a very conservative liveness analysis. As proved in [Battacharyya et al. 1996], the minimum buffer size for which the simple graph $A \xrightarrow{p\quad q} B$ is live is equal to $p + q - \gcd(p, q)$[7]. This however does not imply that any graph whose channels are sized this way is live. Still, this analytic equation is used in many algorithms of buffer sizing to compute a lower bound on buffer sizes as a starting solution ([Stuijk et al. 2006a; Bempelis 2015]). Marchetti and Munier [Marchetti and Kordon 2009] propose a polynomial time symbolic liveness algorithm for Weighted Event Graphs (WEG), a class of Petri Nets equivalent to SDF graphs. This algorithm can be used to minimize the number of initial tokens in a WEG, a problem which is equivalent to minimizing the buffer sizes in an SDF graph.

[7]The equation is slightly different when there are initial tokens.

Let $\vec{s}_i$ denotes the *token timestamp vector*, where each entry corresponds to the production time of tokens in the i^{th} iteration of the graph. Then, as shown in [Geilen 2011], the max-plus algebra can be used to express the evolution of the token timestamp vector: $\vec{s}_i = M\vec{s}_{i-1}$. The eigenvalue of matrix M is equal to the period of the graph. In case of parametric rates, it is sometimes possible to extract a max-plus characterization of the graph with a parametric matrix [Skelin et al. 2014; 2015].

[Ghamarian et al. 2008] presents a parametric throughput analysis for SDF graphs with *bounded* parametric execution times of actors but constant rates. Since rates and delays are non-parametric, the SDF-to-HSDF transformation is possible and the throughput analysis is based on the MCM of the resulting HSDF graph. Therefore, all cycle means are linear functions in terms of the parametric execution times. By using these linear functions, the parameter space is thus divided into a set of convex polyhedra called "throughput regions", each with a throughput expression. [Damavandpeyma et al. 2012] have extended this approach to scenario-aware dataflow (SADF) graphs.

A different analytic approach to estimate lower bounds of the maximum throughput is to compute strictly periodic schedules instead of ASAP schedules (*e.g.,* [Bodin et al. 2013]). This approach is similar to our *Stretch* linearization method used in Section 7 to compute the latency of the graph. We have however shown that using both *Push* and *Stretch* methods usually gives better results. The advantage of the strictly periodic scheduling approach is its capability to handle cyclic graphs. However, not all cyclic graphs have strictly periodic schedules. Furthermore, experiments on real-life benchmarks show that these approaches result in huge over-approximations (sometimes 7 times the exact value) [Bodin et al. 2013].

10. CONCLUSION

We first studied analytically the different cases of the execution of a completely parametric single edge dataflow graph $A \xrightarrow{p \quad q} B$. We introduced enabling patterns to better characterize the data-dependency between the producer and the consumer. We presented the exact symbolic solutions for the minimum buffer size needed by a single edge graph to achieve its maximal throughput. We also presented exact symbolic analyses for computing the latencies of such a graph.

Then, using these results and novel forward linearization techniques, we provided safe upper bounds of buffer sizes of acyclic graphs for maximal throughput. Furthermore, we proposed a heuristic to improve these bounds for graphs with a chain or a tree structure. We also proposed new forward and backward linearization techniques to compute over-approximations of the multi-iteration latency of general acyclic graphs and the input-output latency of chains.

The interested reader can find in [Bouakaz et al. 2016a] several experimental evaluations of these algorithms on both synthetic and real benchmarks. These experiments show, for instance, that our heuristic for buffer computation improves the safe upper bounds by 11.1% in average, over-approximates the exact solutions by 25% in average, and can give the optimal solution for some real applications. Furthermore, our symbolic analyses over-approximate the exact solutions by only 2.5% in case of the multi-iteration latency and 13% in case of the input-output latency.

Future work will concern the extension of these analyses to deal with general (*i.e.,* possibly cyclic) dataflow graphs.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

REFERENCES

Shuvra S. Battacharyya, Edward A. Lee, and Praveen K. Murthy. 1996. *Software synthesis from dataflow graphs*. Kluwer Academic Publishers, Norwell, MA, USA.

Vagelis Bebelis, Pascal Fradet, Alain Girault, and Bruno Lavigueur. 2013. BPDF: a statically analyzable dataflow model with integer and boolean parameters. In *Proc. of the 11th ACM Int. Conf. on Embedded Software*. 3:1–3:10.

Evangelos Bempelis. 2015. *Boolean Parametric Data Flow: Modeling - Analysis - Implementation*. Ph.D. Dissertation. Université Grenoble Alpes.

Bishnupriya Bhattacharya and Shuvra S. Bhattacharyya. 2001. Parameterized dataflow modeling of DSP systems. *IEEE Transactions on Signal Processing* 49, 10 (2001), 2408–2421.

Bruno Bodin, Alix Munier Kordon, and Benoît Dupont de Dinechin. 2013. Periodic schedules for Cyclo-Static Dataflow. In *Proc. of the 11th Symposium on Embedded Systems for Real-time Multimedia*. 105–114.

Adnan Bouakaz, Pascal Fradet, and Alain Girault. 2016a. *Symbolic Analysis of Dataflow Graphs (extended version)*. Technical Report 8742. INRIA.

Adnan Bouakaz, Pascal Fradet, and Alain Girault. 2016b. Symbolic Buffer Sizing for Throughput-Optimal Scheduling of Dataflow Graphs. In *Proc. of Real-Time and Embedded Tech. and Applications Symp.*

Morteza Damavandpeyma, Sander Stuijk, Marc Geilen, Twan Basten, and Henk Corporaal. 2012. Parametric throughput analysis of scenario-aware dataflow graphs. In *Proc. of the Int. Conf. on Computer Design*. 219–226.

Jack B. Dennis. 1974. First version of a data flow procedure language. In *Symp. on Programming*. 362–376.

Karol Desnos, Maxime Pelcat, Jean-François Nezan, Shuvra S Bhattacharyya, and Slaheddine Aridhi. 2013. PiMM: parametrized and interfaced dataflow meta-model for MPSoCs runtime reconfiguration . In *Proc. of Int. Conf. on Embedded Comp. Syst.: Architectures, Modeling, and Simulation*. 41–48.

Pascal Fradet, Alain Girault, and Peter Poplavko. 2012. SPDF: a schedulable parametric data-flow MoC. In *Proc. of Design, Automation and Test in Europe*. 769–774.

Marc Geilen. 2011. Synchronous Dataflow Scenarios. *ACM TECS* 10, 2 (2011), 16:1–16:31.

Amir H. Ghamarian, Marc Geilen, Twan Basten, and Sander Stuijk. 2008. Parametric throughput analysis of synchronous data flow graphs. In *Proc. of Design, Automation and Test in Europe*. 116–121.

Amir H. Ghamarian, Sander Stuijk, Twan Basten, Marc Geilen, and Bart D. Theelen. 2007. Latency Minimization for Synchronous Data Flow Graphs. In *Proc. of the 10th Euromicro Conf. on Digital System Design Architectures, Methods and Tools*. 189–196.

Gilles Kahn. 1974. The semantics of a simple language for parallel programming. In *Information processing*. 471–475.

Akash Kumar, Henk Corporaal, Bart Mesman, and Yajun Ha. 2011. *Multimedia Multiprocessor Systems: Analysis, Design and Management*. Springer Science+Business Media B.V., New York, NY, USA.

Edward A. Lee and David G. Messerschmitt. 1987. Synchronous data flow. In *Proc. of the IEEE*, Vol. 75. 1235–1245.

Olivier Marchetti and Alix Munier Kordon. 2009. A sufficient condition for the liveness of weighted event graphs. *European Journal of Operational Research* 197, 2 (2009), 532–540.

Orlando Moreira, Twan Basten, Marc Geilen, and Sander Stuijk. 2010. Buffer Sizing for Rate-Optimal Single-Rate Data-Flow Scheduling Revisited. *IEEE Trans. Comput.* (2010), 188–201.

Mladen Skelin, Marc Geilen, Francky Catthoor, and Sverre Hendseth. 2014. Worst-case throughput analysis for parametric rate and parametric actor execution time scenario-aware dataflow graphs. In *Proc. of the 1st Int. Workshop on Synthesis of Continuous Parameters*. 65–79.

Mladen Skelin, Marc Geilen, Francky Catthoor, and Sverre Hendseth. 2015. Parametrized dataflow scenarios. In *Proc. of the 12th Int. Conf. on Embedded Software*. 95–104.

Sander Stuijk, Marc Geilen, and Twan Basten. 2006a. Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs. In *Proc. of Design Automation Conf.* 899–904.

Sander Stuijk, Marc Geilen, and Twan Basten. 2006b. SDF^3: SDF for free. In *Proc. of the 6th Int. Conf. on Application of Concurrency to System Design*. 276–278.

Shuvra S. Bhattacharyya Sundararajan Sriram. 2000. *Embedded multiprocessors: scheduling and synchronization*. Marcel Dekker, Inc., New York, NY, USA.

William Thies and Saman P. Amarasinghe. 2010. An empirical characterization of stream programs and its implications for languages and compiler design. In *Proc. of the 19th International Conference on Parallel Architecture and Compilation Techniques*. 365–376.

Online Appendix to: Symbolic Analyses of Dataflow Graphs[8]

ADNAN BOUAKAZ, PASCAL FRADET, and ALAIN GIRAULT, Univ. Grenoble Alpes, Inria, CNRS, LIG, F-38000 Grenoble, France

This electronic appendix contains the most important proofs for the ACM TODAES paper "Symbolic Analyses of Dataflow Graphs". Additional proofs and details (experiments, algorithms, examples, notes, etc.) can be found in a companion paper [Bouakaz et al. 2016a].

A. THROUGHPUT AND DUALITY

PROPERTY 3.1 (THROUGHPUT). *The maximal throughput of an acyclic SDF graph $G = (V, E)$ is equal to*

$$\mathcal{T}_G = \frac{1}{\max\limits_{A \in V}\{z_A t_A\}} \tag{5}$$

Hence, the minimal period is $\mathcal{P}_G = \max\limits_{A\in V}\{z_A t_A\}$.

PROOF. We prove this result by considering the MCM analysis of the corresponding HSDF graph. For a given acyclic SDF graph G, let *HSDF*(G) denote the HSDF graph equivalent to G. The only cycles in *HSDF*(G) are those used to represent the infinite firings of the same actor (see Fig. 5(a)). For each actor A, its corresponding cycle contains one delay and z_A instances of A. Thus, the cycle mean is equal to $z_A t_A$, hence the MCM is equal to $\max\limits_{A\in V} z_A t_A$ and denotes the inverse of the maximal throughput of *HSDF*(G) and G. □

THEOREM 3.1 (DUALITY THEOREM). *Let G be any (possibly cyclic) live SDF graph and G^{-1} be its dual, then $\mathcal{T}_G = \mathcal{T}_{G^{-1}}$ and $\forall i.\ \mathcal{L}_G(i) = \mathcal{L}_{G^{-1}}(i)$.*

PROOF. The detailed proof is in [Bouakaz et al. 2016a]. The first step involves showing that *HSDF*(G) is the dual (after renaming actors) of *HSDF*(G^{-1}). The SDF-to-HSDF transformation algorithm [Sundararajan Sriram 2000] replicates each actor A in the original graph z_A times, each instance representing a firing of A in one iteration (*e.g.,* Fig. 5(a) contains 3 replicas of A), and transforms each edge $A \xrightarrow{p\ \ q} B$ in the original graph, independently of the other edges, into $p\, z_A$ edges, each one representing a data dependency between a firing of A and a firing of B (*e.g.,* Fig. 5(a) contains 6 edges from replicas of A to replicas of B). Hence, it is sufficient to prove that *HSDF*$(A \xrightarrow{p\ \ q} B)$ and *HSDF*$(B \xrightarrow{q\ \ p} A)$ are dual to each other. This is done by showing that, for each data dependency in *HSDF*$(A \xrightarrow{p\ \ q} B)$, there is a dual dependency in *HSDF*$(B \xrightarrow{q\ \ p} A)$.

The second step involves showing that both graphs *HSDF*(G) and *HSDF*(G^{-1}) have the same MCM. By Eq. (5), it follows that $\mathcal{T}_G = \mathcal{T}_{G^{-1}}$. We then prove that $\forall i.\ \mathcal{L}_{HSDF(G)}(i) = \mathcal{L}_{HSDF(G^{-1})}(i)$ by unfolding both HSDF graphs for i iterations and obtaining two dual directed acyclic graphs (DAGs). Therefore, for each maximal path in the first DAG (*i.e.,* a path from a source node to a sink node), there is a dual maximal path in the second DAG. Both such paths have the same length, which concludes the

DOI: 0000001.0000001

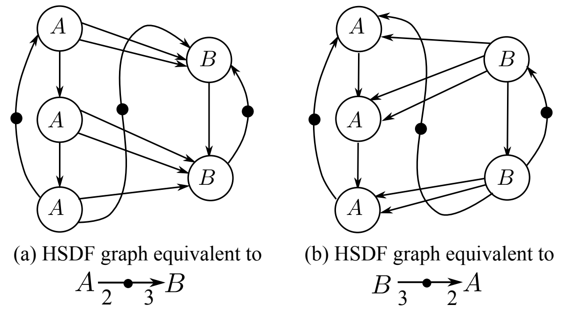

(a) HSDF graph equivalent to $A \xrightarrow[2\quad 3]{\bullet} B$
(b) HSDF graph equivalent to $B \xrightarrow[3\quad 2]{\bullet} A$

Fig. 5. SDF-to-HSDF transformation of a graph and its dual.

proof. This theorem can be also proved using the max-plus algebra and showing the transposition relation between the max-plus characteristic matrices of a graph and its dual. □

B. LINEARIZATION OF $A \xrightarrow{p \quad q} B$

PROPERTY 5.1. *If $z_A t_A < z_B t_B$, then a valid backward lower bound linearization of A is given by*

$$f_{A^\ell}(i) = it_{A^\ell} + (t^0_{A^\ell} + s_B(j_0 + 1) - i_0 t_{A^\ell}) \tag{25}$$

where (i_0, j_0) is a solution of the equation $i_0 p - j_0 q = d$ (d being the smallest buffer size guaranteeing the maximal throughput), and t_{A^ℓ} and $t^0_{A^\ell}$ are the results of the forward lower bound linearization of A in the dual graph G^{-1}.

PROOF. The key element to compute the backward lower bound linearization lies in the following observation, which relates backward linearization to forward linearization. Fig. 14(a) shows the ASAP schedule of graph $G = A \xrightarrow{8 \quad 5} B$ such that $t_A = 5$, $t_B = 6$ and the buffer size is equal to 22. Actor B imposes the highest load. Fig. 14(b) shows the ASAP schedule of the dual graph $G^{-1} = B \xrightarrow{5 \quad 8} A$ with the same buffer size. The producer B in G^{-1} imposes the highest load.

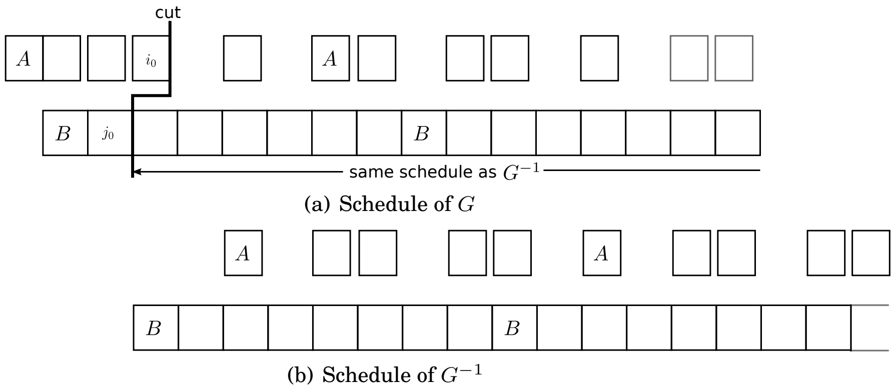


Fig. 14. Relation between forward and backward linearizations for $G = A \xrightarrow{8 \quad 5} B$ with $t_A = 5$, $t_B = 6$ and buffer size equal to 22.

As illustrated in Fig. 14(a), there is an initial phase in the schedule of G, composed of i_0 firings of A and j_0 firings of B, after which the schedule of G is similar to that of G^{-1}. In some cases, the firings of A after the initial phase can be a bit delayed compared to those of A in the dual graph. However, assuming an exact similarity will be an under-approximation of the start times of A, and hence an over-approximation of the input-output latency.

Recall that a buffer of size equal to d is modeled by adding a backward edge with d initial tokens. At the boundary of the initial phase (the "cut" in Fig. 14(a)), the forward edge will contain d tokens while the backward edge will be empty, and this graph coincides exactly with the dual graph G^{-1} in its initial state. The fact that all initial tokens on the backward edge (*i.e.,* d tokens) are transferred to the forward edge is modeled by the following equation:

$$i_0 p - j_0 q = d$$

If the buffer size is a multiple of $\gcd(p, q)$, then the Diophantine equation above is always solvable. If the buffer size is not a multiple of $\gcd(p, q)$, then it can be diminished without affecting the ASAP schedule. In both cases, the computed pair (i_0, j_0) (called the "cut" in Fig. 14) allows to transform the graph $A \to B$ into a dual graph $B \to A$ such that all firings of A after i_0 in the first graph have the same dependencies as the firings of A in the second graph. Hence, if we take $s_B(j_0 + 1)$ as the relative point of origin, the firings of A have the same starting times in both graphs (as can be seen in Fig. 14)

Our linearization of A achieves an under-approximation of the starting times of A, that is an over-approximation of the latency. Let $\tilde{f}_A(i)$ denote the finish time of the i^{th} firing of A in the ASAP schedule of the dual graph G^{-1}. Hence, we have:

$$\forall i \geq 1.\ \tilde{f}_A(i) + s_B(j_0 + 1) \leq f_A(i + i_0)$$

which means that the finish time of the $(i + i_0)^{th}$ firing of A in G can be under-approximated by the finish time of the corresponding i^{th} firing of A in the dual graph, *i.e.,* $\tilde{f}_A(i)$, plus the shift due to the initial phase, *i.e.,* $s_B(j_0 + 1)$.

As described in Section 5.1.2 of the main paper, it is possible to compute a forward lower bound linearization of the firings of A in the dual graph G^{-1} since A is the consumer. We can then put $\tilde{f}_A(i) \geq i t_{A^\ell} + \tilde{t}^0_{A^\ell}$. The following equation is therefore a valid lower bound linearization.

$$\forall i > i_0.\ f_A(i) \geq i t_{A^\ell} + t^0_{A^\ell}$$

where $t^0_{A^\ell} = \tilde{t}^0_{A^\ell} + s_B(j_0 + 1) - i_0 t_{A^\ell}$.

Note that this last equation is also a valid lower bound for all $i \leq i_0$. □

We now prove Eq. (27) below, used to over-approximate the input-output latency of graph $A \xrightarrow{p\quad q} B$ when B imposes a higher load than A:

$$\rho = \frac{d}{q} t_B - t^0_{A^\ell} \tag{27}$$

PROOF. First, consider the dual graph $B \xrightarrow{q\quad p} A$, since $z_B t_B \geq z_A t_A$, Eq. (14) holds, hence, by duality (Theorem 3.2), we have $\mathcal{L}_G(n) = n\mathcal{P}_G + \Delta_{B,A}$. In graph G, $\Delta_{B,A}$ represents the sum of all the initial gaps before the continuous firings of B. Therefore, $s_B(j_0 + 1) = j_0 t_B + \Delta_{B,A}$.

Recall that $\rho = \mathcal{L}_G(n) - f_{A^\ell}(n z_A)$. Since $\mathcal{L}_G(n) = n\mathcal{P}_G + \Delta_{B,A}$ and $f_{A^\ell}(n z_A) = n\mathcal{P}_G + (t^0_{A^\ell} + s_B(j_0 + 1) - i_0 t_{A^\ell})$, we have:

$$\begin{aligned} \rho &= n\mathcal{P}_G + \Delta_{B,A} - n\mathcal{P}_G - (t^0_{A^\ell} + j_0 t_B + \Delta_{B,A} - i_0 t_{A^\ell}) \\ &= i_0 t_{A^\ell} - j_0 t_B - t^0_{A^\ell} \end{aligned}$$

Now, since $t_{A^\ell} = \frac{p}{q} t_B$ and $i_0 \frac{p}{q} - j_0 = \frac{d}{q}$, we finally have $\rho = \frac{d}{q} t_B - t^0_{A^\ell}$. □

C. BUFFER SIZING FOR ACYCLIC GRAPHS

PROPERTY 6.1. *Let G be a graph without any undirected cycle, if the buffer of every channel $A \xrightarrow{p \quad q} B$ in G is at least $\theta^u_{A,B} = 2(p + q - \gcd(p, q))$, then the ASAP execution of the graph achieves the maximal throughput.*

PROOF. We present the proof for chains. The proof for general graphs with no undirected cycle can be found in [Bouakaz et al. 2016a]. Let G be the chain $\{A_1 \xrightarrow{p_1 \quad q_1} A_2 \xrightarrow{p_2 \quad q_2} A_3 \to \cdots \to A_n\}$, according to Eq. (5), the minimal period of G is $\mathcal{P}_G = \max_{i=1..n}\{z_{A_i} t_{A_i}\}$. The period and therefore the throughput remain the same if the execution time of each actor A_i is considered to be $\frac{\mathcal{P}_G}{z_{A_i}}$. Let $G_=$ be the version of G where all actors have the same load as the maximum load in G. Then G and $G_=$ have the same period and throughput.

If the size of each buffer $A_i \xrightarrow{p_i \quad q_i} A_{i+1}$ in $G_=$ is $\theta^u_{A_i,A_{i+1}} = 2(p_i + q_i - \gcd(p_i, q_i))$, then $G_=$ still achieves the maximal throughput. Indeed, size $2(p_1 + q_1 - \gcd(p_1, q_1))$ for the first channel allows both A_1 and A_2 to run consecutively in the steady state (see Eq. (13) in the main paper). Similarly, size $2(p_2 + q_2 - \gcd(p_2, q_2))$ for the second channel allows both A_2 and A_3 to run consecutively, and so on.

Since graph $G_=$ with these buffer sizes achieves the maximal throughput, reducing the execution times of actors in $G_=$ to their original values will never decrease the throughput of the graph thanks to the monotonicity of the self-timed execution. Hence, graph G with these buffer sizes achieves the maximal throughput. □

PROPERTY 6.2. *Let two different chains from A_1 to A_n such that $f_{A^u_n,1}(i) = t_{A_n} i + s_1$, $f_{A^u_n,2}(i) = t_{A_n} i + s_2$ with $s_1 < s_2$, and such that the size of each buffer $A_i \xrightarrow{p_i \quad q_i} A_{i+1}$ is equal to $\theta^u_{A_i,A_{i+1}} = 2(p_i + q_i - \gcd(p_i, q_i))$. In order to prevent the second chain from disturbing the schedule of the first one, it suffices to increase the size of the last channel $A_{n-1} \xrightarrow{p_{n-1} \quad q_{n-1}} A_n$ of the first chain by ζ:*

$$\zeta = \left\lceil \frac{s_2 - s_1}{t_{A_{n-1}}} \right\rceil p_{n-1} \tag{28}$$

PROOF. The proof is based on the following observation. Let $A \xrightarrow{p \quad q} B$ be a graph with two actors such that $z_A t_A = z_B t_B$. Hence, $\theta^u_{A,B} = 2(p+q-\gcd(p, q))$. Even if the first firing of B starts only at time $s_B = \frac{t_B}{q}(p + q - \gcd(p, q))$, as indicated by forward upper bound linearization (Eq. (20) of the main paper), actor A can still fire consecutively provided that the buffer size is $\theta^u_{A,B}$. Hence, if all actors in the first chain are delayed as indicated by the forward upper bound linearization, then the sizes $\theta^u_{A_i,A_{i+1}}$ still allow the first actor A_1 to fire consecutively. Note that after introducing these delays, no actor gets idle once it starts executing.

Let s_1 and s_2 denote the start times of the last actor A_n in the two different chains as computed by the previous process. If $s_2 > s_1$, then the extra delay $(s_2 - s_1)$ imposed by the second chain may prevent A_1 from running consecutively. Let $A_{n-1} \xrightarrow{p_{n-1} \quad q_{n-1}} A_n$ be the last channel in the first chain. Increasing the size of this channel by $\left\lceil \frac{s_2-s_1}{t_{A_{n-1}}} \right\rceil p_{n-1}$, as explained in the previous example, will allow actor A_{n-1} to fire consecutively during the extra delay $(s_2 - s_1)$. That is, the impact of the second chain on the first one is avoided. □

D. LATENCY COMPUTATION FOR ACYCLIC GRAPHS

PROPERTY 7.1. *For any acyclic SDF graph* G*,* $\forall i.\ \mathcal{L}_G(i) = \max_{g \in \mathcal{G}(G)}\{\mathcal{L}_g(i)\}$

PROOF. This property follows immediately form the compositionality of the SDF-to-HSDF transformation, *i.e., HSDF*$(G) = \bigcup_{g \in \mathcal{G}(G)}$ *HSDF*(g). Therefore, any maximal path in the DAG obtained by unfolding *HSDF*(G) for i iterations will be found in the DAG obtained by unfolding some graph $g \in \mathcal{G}(G)$ for i iterations. Indeed, since G does not contain any cycle (except self-edges), there will be no path from actor A to B in *HSDF*(G) unless both actors belong to the same chain. □