HAL
open science

Symbolic Analysis of Dataflow Graphs (Extended Version)

Adnan Bouakaz, Pascal Fradet, Alain Girault

To cite this version:

Adnan Bouakaz, Pascal Fradet, Alain Girault. Symbolic Analysis of Dataflow Graphs (Extended Version). [Research Report] 8742, Inria - Research Centre Grenoble – Rhône-Alpes. 2016. hal-01166360v3

HAL Id: hal-01166360

https://inria.hal.science/hal-01166360v3

Submitted on 6 Jan 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Symbolic Analysis of Dataflow Graphs (Extended Version)

Adnan Bouakaz, Pascal Fradet, Alain Girault

RESEARCH REPORT
N° 8742
January 2016
Project-Team Spades

ISSN 0249-6399 ISRN INRIA/RR--8742--FR+ENG

Symbolic Analysis of Dataflow Graphs (Extended Version)

Adnan Bouakaz, Pascal Fradet, Alain Girault

Project-Team Spades

Research Report n° 8742 — version 2 — initial version January 2016 — revised version January 2016 — 60 pages

Abstract: The synchronous dataflow model is widely used to design embedded stream-processing applications under strict quality-of-service requirements (*e.g.,* buffering memory, 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, they may cause a huge run-time overhead or make design space exploration unacceptably slow due to their exponential time complexity.
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 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 simple graph made of a single parametric edge. The results are then extended to general acyclic graphs using linear approximation techniques. We assess the proposed analyses experimentally on both synthetic and real benchmarks.

Key-words: Synchronous dataflow graphs, as soon as possible scheduling, buffer minimization, throughput, latency, symbolic analysis

RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Analyse Symbolique des Graphes Flots de Données (Version Étendue)

Résumé : Les graphes flots de données sont largement utilisés pour modéliser les applications de traitement de signal et de streaming. Ces modèles permettent de prévoir et garantir les performances (*e.g.,* débit, tailles des buffers, latence) de ces applications. L'analyse de performance peut être effectuée au moment de compilation (exploration de l'espace de conception) ou au moment d'exécution (gestion de ressources, systèmes adaptables). Cependant, ces algorithmes ont souvent une complexité exponentielle, et peuvent donc introduire une pénalité significative au moment d'exécution ou rendre l'exploration de l'espace de conception excessivement lente. Nous montrons dans ce papier que les analyses symboliques sont mieux adaptées dans ce contexte car elles expriment la performance du système par des fonctions en termes de paramètres (*i.e.,* taux de production et de consommation, temps d'exécution). Telles fonctions peuvent être rapidement réévaluées pour tout changement de configuration du système ou utilisées pour vérifier la satisfiabilité des besoins non fonctionnels.

Nous proposons des analyses symboliques pour calculer le débit maximale des graphes acycliques, les tailles de buffers minimales requise pour réaliser ce débit, et la latence entrée/sortie. Nous effectuons d'abord une étude analytique et exacte dans le cas d'un graphe avec seulement deux acteurs. En suite, nous étendons ces résultats pour les graphes acycliques. Enfin, nous évaluons notre approche en utilisant des cas d'études réels et des benchmarks synthétiques.

Mots-clés : graphes flots de données, débit, tailles minimales des buffers, latence, analyse symbolique

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 [14]. 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) [7] and Kahn process networks (KPN) [13] allow to explicitly express parallelism and communications where tasks (or actors) are independent and communicate only through channels. Using a dataflow model, 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 input sequence, the system will always produce the same output results. 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 [15] 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 [4], SPDF [9], BPDF [2], πSDF [8], *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 the timing requirements, and to use dynamic voltage and frequency scaling (DVFS) techniques to reduce energy consumption in case the maximum throughput is larger than the desired quality-of-service [17, 21]. Latency is another important timing constraint that is usually used in the design of real-time control systems. It measures the time delay between the stimulation and response, and hence the reactiveness of the system and its ability to deliver a regular and timely response. Most embedded systems must comply to severe constraints on the size, weight, power and cost. Therefore, minimization of memory requirements is a primordial step in the design of such systems. These three measures (throughput, latency and memory) are often antagonistic. A huge effort have been devoted in the past decades to solve these problems.

In this paper, we focus on self-timed scheduling which 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 of parameters, 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 the application to achieve its maximum throughput?

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

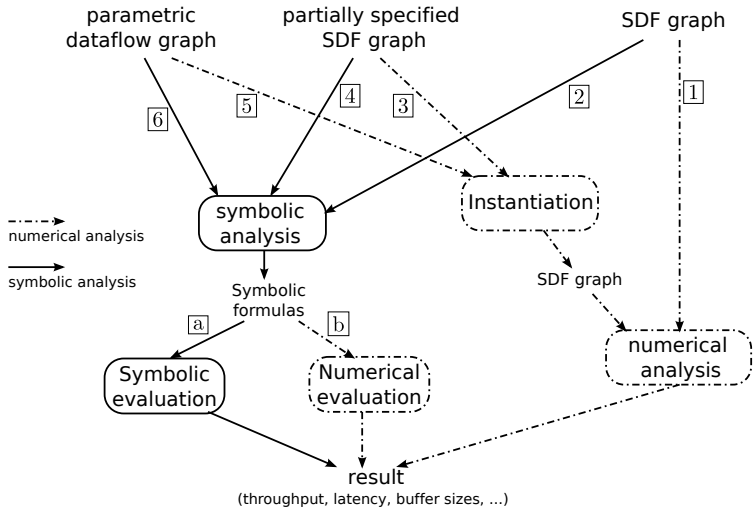


Figure 1: Symbolic and numerical analyses.

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

1. At early design stages, the SDF model of the application is usually partially specified and design space exploration may require analyzing of a potentially huge number of configurations (path 3). Symbolic analyses are a big advantage in this case: formulas are generated only once and simply evaluated for each possible configuration (*i.e.,* set of parameters) (path 4). Indeed, exact algorithms for throughput and latency computation may be acceptable at compile-time in practice; however, frequent calls to these algorithms to check a large set of configuration values make design space exploration unacceptably slow.

2. 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 6b) or, better, by an *analytic proof* (path 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.

3. For dynamic models and run-time resource management, appropriate settings have sometimes to be chosen dynamically. Consider, for instance, a parametric application where frequency scaling is used to guarantee a specific throughput and to minimize power consumption. In such case, frequency must be adjusted at each parameter change. Instantiating the graph (path 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 6b).

4. Finally, even for completely static SDF models, many analyses have an exponential complexity. Indeed, exact algorithms for minimal buffer sizes are too expensive even for small graphs. [16] shows that this problem is NP-complete for homogeneous SDF (HSDF) graphs. Moreover, 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.

The article is organized as follows. Section 2 introduces the application model, the scheduling policy, and the needed definitions. Section 3 presents the throughput analysis of acyclic SDF graphs and a new generic result (the duality theorem) required to solve the other questions. We present in Section 4 different symbolic analyses for a single SDF edge $A \xrightarrow{p\ \ q} B$. Section 5 describes linearization techniques for graph $A \xrightarrow{p\ \ q} B$ that are used in Sections 6 and 7 to extend the results of Section 4 to general acyclic graphs. The proposed algorithms are evaluated on both synthetic and real benchmarks in Section 8. Finally, we review related work in Section 9 and conclude in Section 10. Proofs of all properties and theorems can be found in the appendix which has the same structure as the paper for a better readability.

2 Background

In this section, we first introduce the application model and the scheduling policy. Then, we review some useful definitions and properties.

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 a given actor (called *firing*) starts by consuming data tokens from all its incoming edges (its *inputs*), computes and finishes by producing data tokens to all its outgoing edges (its *outputs*). The number of tokens

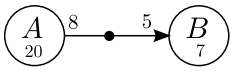


Figure 2: A simple SDF graph

consumed (resp. produced) at a given input (resp. output) edge at each firing is 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* (also called *delays*). In the following, we denote the execution time of an actor X by t_X. For instance, Fig. 2 shows an SDF graph with two actors A and B, with execution times $t_A = 20$ and $t_B = 7$ respectively. The production and consumption rates of channel $A \to B$ are 8 and 5 respectively. This edge carries one initial token (represented by the black dot).

The *state* of a dataflow graph is the vector of the number of tokens present at each edge (*i.e.,* buffered in each FIFO channel). Each edge carries zero or more tokens at any moment. The *initial state* of a graph is specified by the number of initial tokens on its edges. The 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* [15]. We write 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 $A \xrightarrow{p\ \ q} B$ 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.

In this paper, we study only *consistent acyclic* SDF graphs with initially empty channels. Inconsistent graphs are of less importance since they cannot be implemented with bounded memory without deadlocking.

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*) their throughput can be computed as the inverse of the *maximal cycle mean* (MCM) of the graph; and, (*ii*) any consistent SDF graph can be converted into an HSDF graph. The cycle mean of a cycle is equal to the sum of execution times of the actors in the cycle divided by the number of delays (*i.e.,* initial tokens) in the channels of this cycle. This provides a way to compute the throughput of any SDF graph. There are two drawbacks in this approach: first, the translation from SDF to HSDF may lead 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 [20], 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 *prologue* followed by a *steady state* that repeats infinitely. Fig. 3 shows an ASAP schedule where thick broken lines represent the iterations boundaries.

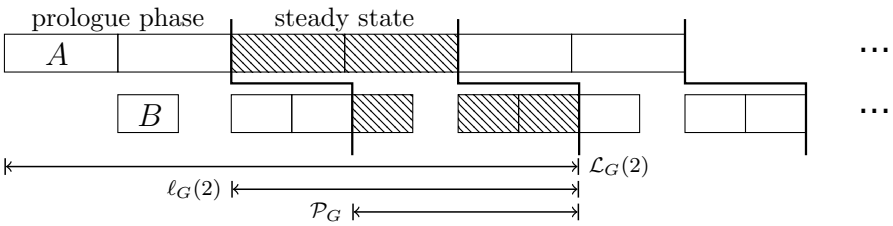


Figure 3: ASAP schedule of graph $A \xrightarrow{3\ \ 2} B$ with t_A=15 and t_B=8

Fig. 4 shows how to make the absence of auto-concurrency explicit in an SDF graph by adding self-edges with rates equal to 1 and one initial token. A firing of actor A will consume 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.

In the SDF model, channels are unbounded. However, the size of a channel $A \xrightarrow{p\ \ q} B$ can be constrained to d tokens by adding a backward channel $B \xrightarrow{q\ \ p} A$ 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

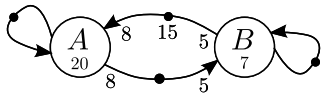


Figure 4: The SDF graph of Fig. 2 with channel size constraint and auto-concurrency disabled

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. This is illustrated in Fig. 5 where an actor X is represented by different actors for reading inputs (X_r), computing (X_c) and writing results (X_w). For instance, A_w produces 8 tokens (and checks for 8 places) only after the computation (A_c); B_r frees 5 buffer places before the computation (B_c) starts. Note that only actors representing computation have a non-null execution time. Actually, the graph in Fig. 2 requires a minimum buffer size of 20

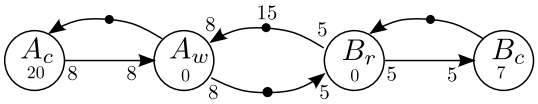


Figure 5: Less conservative modeling of channel size constraint on the graph of Fig. 2

to achieve the maximal throughput when using the first modeling technique (Fig. 4) whereas it only needs 12 when using the second (Fig. 5). This shows that different modeling techniques may lead to different symbolic formulas. However, the approach proposed in this paper for symbolic computation of buffer sizes can be adapted to any modeling technique.

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 (assuming timing starts at 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 relation is particularly useful in the case of parametric dynamic dataflow models where parameter reconfigurations are frequent. Hence, if a given configuration lasts only during m iterations, then $m/\mathcal{L}_G(m)$ gives the achievable throughput for the current configuration. The designer can use this information, for instance, to reduce the frequency of processors and save energy as long as the desired quality-of-service is guaranteed.

The *input-output latency* $\ell_G(n)$ of the nth 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 nth iteration. The definition given in [12] 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 in case of real-time control systems since it specifies the maximum delay between sampling data from sensors and sending control commands to the actuators.

For a channel $A \xrightarrow{p\ \ q} B$ with d initial tokens, the ith firing of B (denoted B_i) is enabled if and only if the number of produced tokens is larger than $i\,q$. Hence, B has to wait for the jth firing of A (denoted A_j) such that $j\,p + d \geq iq$. The *data-dependency* between A and B is formalized by the following equation.

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

The ceiling function 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 proerty on dual graphs that is 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*

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

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

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 which has the highest load; *i.e.,* actor $\text{argmax}_{A \in V}\{z_A t_A\}$. This implies that this actor never gets idle 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 3.1 (Duality theorem)**.** *Let G be any (cyclic or not) live SDF graph and G^{-1} be its dual, then $\mathcal{T}_G = \mathcal{T}_{G^{-1}}$ and $\forall i.\ \mathcal{L}_G(i) = \mathcal{L}_{G^{-1}}(i)$.*

Fig. 6 illustrates the duality theorem with the SDF graph G_a of Fig. 4. The latency of the first iteration of the ASAP execution of that graph is equal to the latency of the first iteration of its dual *i.e.,* $\mathcal{L}_{G_a}(1) = \mathcal{L}_{G_a{}^{-1}}(1)$.

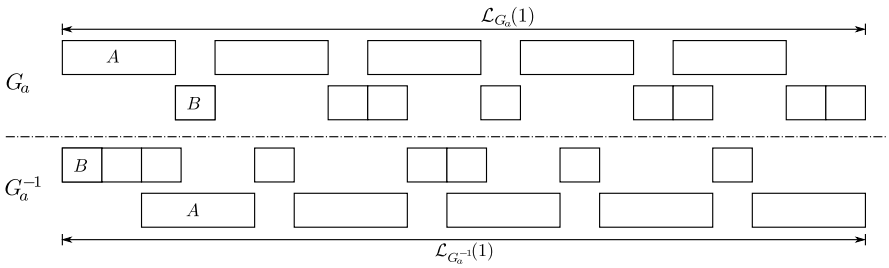


Figure 6: Illustration of the duality theorem.

Note 3.1. The ALAP (As Late As Possible) schedule of graph G is identical to the reversed ASAP schedule of its dual G^{-1}. In other terms, the ALAP schedule of G in the time interval $[0, n]$ is identical to the ASAP schedule of G^{-1} starting at n and ending at 0. This provides a second method to prove the first part of the duality theorem; *i.e.,* $\mathcal{T}_G = \mathcal{T}_{G^{-1}}$. Indeed, [10] states that both the ASAP schedule and the ALAP schedule of an SDF graph have the same throughput.

We use the transformation of a graph to its dual as well as the associated theorem 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 parameric acyclic SDF graph made of a single edge: $G = A \xrightarrow{p \quad q} B$. We provide exact symbolic formulas for minimum buffer size and latency questions. The graph G is parametrized by the production and consumption rates $p, q \in \mathbb{N}^+$ as well as the execution times $t_A, t_B \in \mathbb{R}^+$. This section shows that the symbolic analysis, even for such simple graphs, is quite involved.

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:

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

4.1 Enabling patterns

We introduce *enabling patterns*, which characterize the data-dependency between a producer and a consumer. Compared to Eq. (4), they are more intuitive and suitable to the reasoning about buffer sizes and latency.

Enabling patterns between the producer A and consumer B are defined by the following grammar:

$$P \quad ::= \quad A^i \rightsquigarrow B^j \quad | \quad [P]^{x=1..k} \quad | \quad P_1; P_2$$

where i, j, k evaluate to positive integers.

An enabling pattern P is either a basic pattern ($A^i \rightsquigarrow B^j$), a repetition for k times ($[P]^{x=1..k}$), or a sequence of enabling patterns $P_1; P_2$, The expressions i, j or k are arithmetic expressions made of integers, parameters or pattern variables defined by enclosing repetition patterns.

The semantics of an enabling pattern between actors A and B is defined *w.r.t.* two counters a and b representing the number of completed firings of A and B (initially 0). The pattern $A^i \rightsquigarrow B^j; P$ *w.r.t.* (a, b) means that:

- a firings of A have produced enough tokens to fire actor B b times and not more;
- then, if A is not fired at least i times more, then B cannot be fired; otherwise B can be fired j times and not more;
- the subsequent pattern P is considered with the new values $(a + i, b + j)$.

Formally

$$\frac{i \geq 1 \quad j \geq 1 \quad ap \geq bq \quad (a+i-1)p < (b+1)q \quad (b+j)q \leq (a+i)p < (b+j+1)q}{(a, b, A^i \rightsquigarrow B^j) = (a+i, b+j)}$$

A repetition $[P]^{x=1..k}$ is a sequence of k patterns P. The pattern $[P]^{x=1..k}$ is also written $[P]^k$ if the pattern variable x is not used in P.

A correct enabling pattern describes an entire iteration, that is, at the end of the pattern, we should have $a = z_A$ and $b = z_B$.

For example, the enabling pattern of $A \xrightarrow{3\ \ 6} B$ is $A^2 \rightsquigarrow B$; *i.e.,* after every two firings of actor A, one firing of B is enabled (B^1 is written B). The enabling pattern of $A \xrightarrow{8\ \ 5} B$ is:

$$A \rightsquigarrow B; A \rightsquigarrow B^2; A \rightsquigarrow B; [A \rightsquigarrow B^2]^2$$

which is illustrated in Fig. 7. This pattern can also be written as the factorized pattern:

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

This factorized representation is particularly useful when the length and shape of enabling patterns depend on parameters.

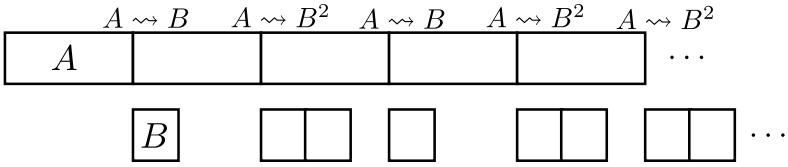


Figure 7: An ASAP execution of $A \xrightarrow{8\ \ 5} B$ with $t_A = 20$ and $t_B = 7$.

Depending on the production and consumption rates p and q, there are six possible enabling patterns.

Property 4.1. *Fig. 8 gathers all possible enabling patterns for the graph* $A \xrightarrow{p\ \ q} B$.

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

We now use enabling patterns to compute the minimum size of the buffer $A \xrightarrow{p\ \ q} B$ (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 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 $\lambda_j = \left\lceil \frac{jr}{p-r} \right\rceil - \left\lceil \frac{(j-1)r}{p-r} \right\rceil$.

Figure 8: Enabling patterns.

• **Case** $z_A t_A \geq z_B t_B$ (*i.e.,* $qt_A \geq pt_B$): 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 jth firing of A can occur immediately after the $(j-1)^{th}$ firing of A. By definition of $\theta_{A,B}$, we have $\theta_{A,B} = \max_j \delta_j$. Let x_j denote the number of firings of B that have finished by the start of the jth 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 to perform the first two firings of A consecutively, the backward edge should have at least $2p$ tokens. Futhermore, since $qt_A \geq kqt_B$ (hence $t_A \geq kt_B$), the k firings of B complete before the third firing of A which still needs $2p$ initial tokens (*i.e.,* $\delta_3 = 2p$) in order to fire again immediately. Hence, the minimum buffer size is $2p$. However, unlike sequence (x_j), enabling patterns are time-independent. Thus, when considering execution times t_A and t_B, three cases will emerge (see I., II. and III. in Fig. 9); 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 the appendix (Section B.2). 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 $z_A t_A \geq z_B t_B$, the minimum buffer sizes of $A \xrightarrow{p \quad q} B$ for maximal throughput are given by the symbolic formulas of Fig. 9.*

• **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 all firings of B cannot be consecutive since initially,

Case I.

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

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

Case I.2. $\text{B.1} \vee ((\text{B.2} \vee \text{B.3}) \wedge (t_B \leq kt_A))$

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

Case II.

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

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

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

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

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 $r' = t_A - kt_B$.

Cases III.(A.3), III.(B.2), III.(B.3) see the proof in the appendix.

Figure 9: Minimum buffer size $\theta_{A,B}$ when $z_A t_A \geq z_B t_B$.

there are no tokens to be consumed. The previous approach can still be followed using 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 $\theta_{A,B} = \theta_{B,A}$.*

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. 9 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.

Property 4.4. *If If $z_A t_A \geq z_B t_B$ and the channel $A \xrightarrow{p\ \ q} B$ contains d initial tokens, then the minimum buffer size $\theta'_{A,B}$ that allows the maximum throughput is*

$$\theta'_{A,B} = \max\{0, \theta_{A,B} - d + d \bmod \gcd(p,q)\} \tag{14}$$

with $\theta_{A,B}$ as defined in Fig. 9.

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

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 $A \xrightarrow{p\ \ q} B$. Since we use the multi-iteration latency to approximate the maximal achievable throughput, we will suppose that buffers are unbounded. There are two cases depending on whether A or B imposes the highest load.

• **Case** $z_A t_A \geq z_B t_B$, *i.e.,* A imposes a higher load than B. As illustrated in Fig. 10, actor A never gets idle and $\mathcal{P}_G = z_A t_A$. Therefore, we can put

$$\mathcal{L}_G(n) = n\mathcal{P}_G + \Delta_{A,B} \tag{15}$$

such that $\Delta_{A,B}$ is the remaining execution time for actor B after actor A has finished its firings of the nth iteration ($\Delta_{A,B}$ is constant over all iterations). The value of $\Delta_{A,B}$ is given by Property 4.5. Eq.(15) shows that the multi-iteration latency will under-approximate the maximal throughput by only a small amount that decreases with the value of n; *i.e.,*

$$\hat{\mathcal{T}}_G = \frac{n}{\mathcal{L}_G(n)} = \frac{n}{n\mathcal{P}_G + \Delta_{A,B}} \leq \mathcal{T}_G = \frac{1}{\mathcal{P}_G}$$

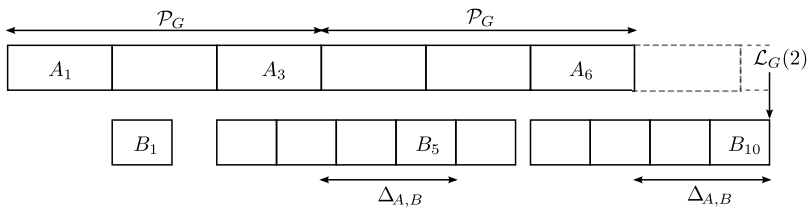


Figure 10: Multi-iteration latency $\mathcal{L}_G(2)$, case $z_A t_A \geq z_B t_B$ (p=5, q=3, t_A=14, t_B=8).

Property 4.5. *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. 11.*

• **Case** $z_A t_A < z_B t_B$, *i.e.,* B imposes a higher load than A. As illustrated in Fig. 12, actor B never gets idle in the steady state. However, in general all firings of B cannot be consecutive since initially, there are no tokens on the forward edge $A \rightarrow B$. Note that $\Delta_{A,B}$ is not constant over all iterations and diverges to infinity if the buffer is supposed unbounded. A simpler way to compute $\mathcal{L}_G(n)$ is to use 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 $\Delta_{B,A}$ can be easily computed using Property 4.5.

Case I.

$$\Delta_{A,B} = \left\lceil \frac{p}{q} \right\rceil t_B \tag{16}$$

Case II.

Case II.1.

$$\Delta_{A,B} = t_A + \left\lceil \frac{r}{q-r} \right\rceil ((k+1)t_B - t_A) \tag{17}$$

Case II.2.

$$\Delta_{A,B} = t_B + \left\lceil \frac{p-r}{r} \right\rceil (t_B - kt_A) \tag{18}$$

Case III.

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

$$\Delta_{A,B} = t_A + r' + \frac{t_B r - qr'}{\gcd(p,q)} + (t_B - r') \max_{j=0}^{n-1} \left(\frac{jr'}{t_B - r'} - \left\lfloor \frac{jr}{q-r} \right\rfloor \right) \tag{19}$$

Cases III.(A.3), III.(B.2), III.(B.3): see the proof in the appendix.

Figure 11: Multi-iteration latency: value of $\Delta_{A,B}$.

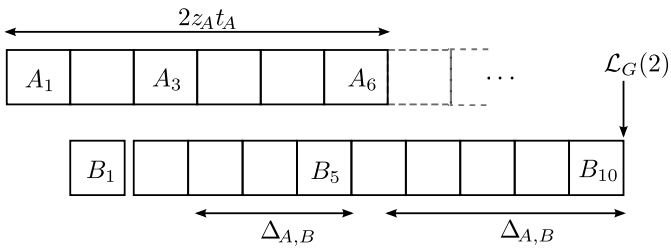


Figure 12: Multi-iteration latency $\mathcal{L}_G(2)$, case $z_A t_A < z_B t_B$ ($p=5$, $q=3$, $t_A=14$,$t_B=12$).

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

In this section, we 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 A or B imposes the highest load.

• **Case** A imposes a higher load than B: The input-output latency is equal to the finish time of the nth iteration, which is equal to $\mathcal{L}_G(n) = n\mathcal{P}_G + \Delta_{A,B}$ (Eq. (15)), minus the start time of the first firing of A in the nth iteration. This start time is equal to $(n-1)\mathcal{P}_G$ since A never gets idle. Therefore, we have

$$\ell_G(n) = \mathcal{L}_G(n) - (n-1)\mathcal{P}_G = \mathcal{P}_G + \Delta_{A,B} \tag{20}$$

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 a higher load than A: We have $\ell_G(n) = \mathcal{L}_G(n) - (n-1)z_A t_A$ 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, which uses a linearization technique presented in Section 5.2.

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 make use of 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. 13, 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 ith 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 fire consecutively as many times as B and such that

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

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. 13 that both the 5th firings of B and B^s finish at the same time.

5.1.1 Upper bound linearization

We present two linearization methods, *Push* and *Stretch*, illustrated in Fig. 13. *Push* considers the actor B^p which is obtained by pushing *all* firings of B to the right end to get rid of all the gaps. This method is suitable only for a finite number of iterations, say n. The execution time remains the same (*i.e.,* $t_{B^p} = t_B$) but the starting time of the consecutive firings is equal to the multi-iteration latency $\mathcal{L}_G(n)$ minus the execution time of nz_B firings: $t^0_{B^p} = \mathcal{L}_G(n) - nz_B t_B$.

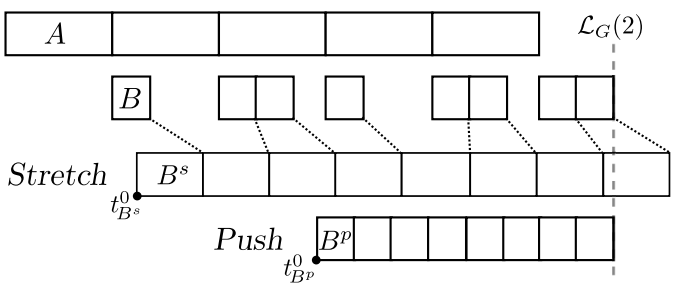


Figure 13: Upper bound linearization ($p = 8$, $q = 5$, $t_A = 20$, $t_B = 7$).

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

• **Case** $z_A t_A \geq z_B t_B$: This case is illustrated in Fig. 13. We have $t_{B^s} = \frac{q t_A}{p}$, *i.e.,* both A and B^s have the same load. The starting time $t^0_{B^s}$ is computed as follows. We have $f_{B^s}(i) = i t_{B^s} + t^0_{B^s} \geq f_B(i)$. Hence, $t^0_{B^s} = \max_i (f_B(i) - i t_{B^s})$. 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 appendix C.1) that $t^0_{B^s} = t_A + t_B - \frac{\gcd(p,q)}{p} t_A$. We conclude that:

$$\forall i.\ f_{B^s}(i) = \frac{q t_A}{p} i + \left(t_A + t_B - \frac{\gcd(p,q)}{p} t_A \right) \tag{21}$$

Method *Stretch* may advance the starting of some firings (*e.g.,* the 2th firing of B^s in Fig. 13), but always postpone their endings.

• **Case** $z_A t_A < z_B t_B$: 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 $\mathcal{L}_G(n) = \mathcal{L}_{G^{-1}}(n) = n z_B t_B + \Delta_{B,A}$ and we can take $t^0_{B^s} = \Delta_{B,A}$, where $\Delta_{B,A}$ is computed on the dual graph (see Property 4.5).

5.1.2 Lower bound linearization

For lower bound linearization, we use the *Stretch* method; *i.e.,* the execution time of B is increased in order to fill the gaps over an infinite execution such that $\forall i.\ f_{B^\ell}(i) = i t_{B^\ell} + t^0_{B^\ell} \leq f_B(i)$. 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. 14 shows the symbolic formulas for $t^0_{B^\ell}$. The reader may refer appendix C.1 for details.

Case A. $p \geq q$

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

$$t^0_{B^\ell} = k t_B + \min \left\{ t_B, \frac{r t_A}{p} \right\} \tag{22}$$

Case B. $p < q$

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

Case I.

$$t^0_{B^\ell} = t_B \tag{23}$$

Cases II + III.

$$t^0_{B^\ell} = t_B + \frac{p-r}{p} t_A + \left\lfloor \frac{p}{r} \right\rfloor \frac{\sigma}{p} + \min \left\{ \frac{-\sigma}{p}, \frac{r - (p \bmod r)}{p} t_A \right\} \tag{24}$$

Figure 14: Linear 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. Hence, $t_{B^\ell} = t_B$ and $t^0_{B^\ell}$ is equal to the start time of the first firing of B.

Thanks to this linearizations, a chain $A \xrightarrow{p\ \ q} B \xrightarrow{p'\ \ q'} C$ can be treated by first scheduling the subgraph $A \xrightarrow{p\ \ q} B$, then linearizing the firings of B if they are not consecutive, 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 a chain $A \to B \to C$, instead of trying to combine the formulas of Fig. 9, and hence solving very complicated combinations.

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

In this section, we propose a backward lower bound linearization of the producer, which is needed to compute the input-output latency of chains. The previous section describes a forward lower bound linearization; *i.e.,* a transformation of the firings of the consumer and not those of the producer.

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). A safe buffer size will be $\theta_{A,B}$ as described in Section 4.2.

Let $s_X(i)$ denote the start time of the ith firing of an 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 input-output latency. Therefore, the required linearization constraint is rather

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

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 equation $i_0 p - j_0 q = d$ (d is the buffer size), 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}.

We show now how to use this property to compute an upper bound of the input-output latency of graph $A \xrightarrow{p \quad q} B$ in case B imposes a higher load than A. Fig. 15 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 over the values of n. The maximum input-output latency can therefore be given by

$$\hat{\ell}_G = \mathcal{P}_G + \rho \tag{26}$$

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

$$\rho = i_0 t_{A^\ell} - j_0 t_B - t^0_{A^\ell} = \frac{d}{q} t_B - t^0_{A^\ell} \tag{27}$$

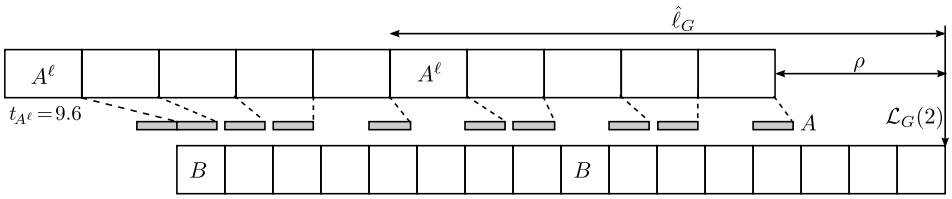


Figure 15: Backward lower bound linearization.

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 acyclic graphs, 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 $A \xrightarrow{p\ \ q} B$ 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 the next section. A simple counterexample is the graph $G_b = A \xrightarrow{3\ \ 4} B \xrightarrow{4\ \ 2} C$ with $t_A = 16$, $t_B = 11$ and $t_C = 12$. The repetition vector is $\vec{z} = [4, 3, 6]$. Actor C imposes the higher load, hence the minimal period of this graph is $\mathcal{P}_{G_b} = z_C t_C = 72$. We have $\theta_{A,B} = 9$ and $\theta_{B,C} = 6$. Locally, these buffer sizes allow the producers to run freely without any constraint from the consumers. However, when they are put together (*i.e.,* the global execution), the maximal throughput, where actor C fires consecutively, cannot be achieved (see Fig. 16). The computation of $\theta_{B,C}$ assumes that the execution time of actor B is $t_B = 11$. However, as illustrated in Fig. 16, there are gaps between the firings of B due to the data-dependency $A \to B$. The global execution proceeds as if the execution times of B were sometimes longer than 11.

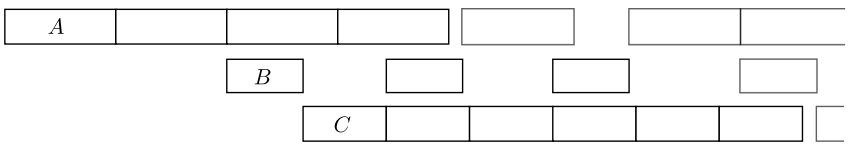


Figure 16: ASAP execution of the graph $G_b = A \xrightarrow{3\ \ 4} B \xrightarrow{4\ \ 2} C$.

Property 6.1. *Let G be a graph without any undirected cycle, if the buffer of every channel $A \xrightarrow{p\ \ q} B$ 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.*

Note 6.1. Since the minimum buffer sizes below which the graph is definitely not live are equal to $p + q - \gcd(p, q)$ [1], 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 $G_c = \{A \xrightarrow{4\ \ 3} B \xrightarrow{3\ \ 8} D,\ A \xrightarrow{1\ \ 3} C \xrightarrow{3\ \ 2} D\}$ 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. 17. 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 after 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 (*i.e.,* by the second chain), which delays the 7th firing of B, which in turn delays the 8th firing of A. Let $f^u_{D,1}$ (resp. $f^u_{D,2}$) denote the 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 $\lceil \frac{28-16}{3} \rceil \times 3 = 12$.

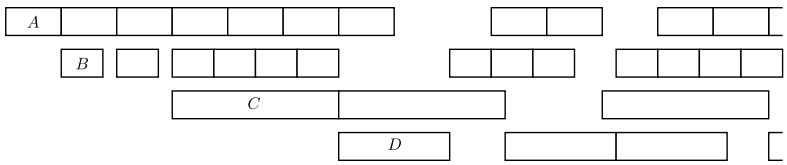


Figure 17: ASAP execution of G_c

In the general case, Eq.(60) gives the value of s_{A_n} for a chain $A_1 \to A_2 \to \ldots \to A_n$ where all actors have the same load. If the actors do not have the same load, we consider, as in the proof of Property 6.1, the chain where all actors have the same load *i.e.,* the maximum load of the original chain.

Property 6.2. *Let two different chains from A_1 to A_n such that $f_{A^u_n,1}(i) = t_{A_n} i + s_1$, $f_{A^u_n,2}(i) = t_{A_n} i + s_2$ and $s_1 < s_2$. To prevent the second chain from disturbing the schedule of the first one, it suffices to increase the size of the last channel $A_{n-1} \xrightarrow{p\ \ q} A_n$ of the first chain by ζ with*

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

This approach can be extended to deal with 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).

Note 6.2. The value of ζ, like the value of $\theta^u_{A,B}$, does not actually depend on execution times. Indeed, Eq. (60) shows that s_1 and s_2 can be expressed as $z_{A_n} t_{A_n} k_1$ and $z_{A_n} t_{A_n} k_2$. Since $z_{A_n} t_{A_n} = z_{A_{n-1}} t_{A_{n-1}}$, term $\frac{s_2 - s_1}{t_{A_{n-1}}}$ can be expressed as $z_{A_{n-1}}(k_2 - k_1)$, which does not depend on execution times.

ALGORITHM 1: Safe upper bounds for graphs with undirected cycles

Input: SDF graph with undirected cycles, all actors impose the same load.
Output: Safe buffer sizes.
L: list of actors in a 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* $A \xrightarrow{p\ \ q} B \in pred(B)$ **do** $s_{B,A} = s_A + \frac{t_B}{q}(p + q - \gcd(p,q))$;
    $s_B = \max\limits_{A \xrightarrow{p\ \ q} B \in pred(B)} (s_{B,A})$;
    for *each edge* $A \xrightarrow{p\ \ q} B \in pred(B)$ **do** $size(A \xrightarrow{p\ \ q} B) = \theta^u_{A,B} + \left\lceil \frac{s_B - s_A}{t_{A_{n-1}}} \right\rceil p$;

Note 6.3. There is a second more precise but more complicated method to compute the size of the last channel $A_{n-1} \xrightarrow{p\ \ q} A_n$ in Property 6.2. Consider a single edge graph $A_{n-1} \xrightarrow{p\ \ q} A_n$ where A_{n-1} and A_n impose the same load. Hence, $\theta_{A_{n-1},A_n} = 2(p + q - \gcd(p,q))$ is the minimum buffer size that allows the ASAP execution to achieve the maximal throughput. As indicated in the proof of Property 6.1, the same size allows both actors to fire consecutively once they start firing, as long as the start time of A_n is less than or equal to $s_{A_n} = \frac{t_{A_n}}{q}(p + q - \gcd(p,q))$. Let $\theta_{A_{n-1},A_n}(x)$ denote the minimum buffer size for a maximal throughput where firings of A_n can start only after x unit of times. We are interested in computing $\theta_{A_{n-1},A_n}(s_{A_n} + (s_2 - s_1))$, *i.e.,* the impact of the extra delay on the buffer size. Following the same approach used to deduce Eq. (59), we have

$$\theta_{A_{n-1},A_n}(s_{A_n} + (s_2 - s_1)) = \max_j \left(jp - q \left\lfloor \frac{jp - (2p + q - \gcd(p,q))}{q} - \frac{s_2 - s_1}{t_{A_n}} \right\rfloor_0 \right)$$

This can be over-approximated by $2p + 2q - \gcd(p,q) + \left\lfloor \frac{(s_2 - s_1)p}{t_{A_{n-1}}} \right\rfloor$. So, the value of ζ in this case is equal to $\left\lfloor \frac{(s_2 - s_1)p}{t_{A_{n-1}}} \right\rfloor + \gcd(p,q)$.

Note 6.4. There are different ways to solve the problem in Property 6.2. If $s_2 > s_1$, then the start time of actor A_n in the combined schedule of the two chains is equal to $\max(s_2, s_1) = s_2$. Suppose that, compared to the linear schedule of the first chain $A_1 \xrightarrow{p_1\ \ q_1} A_2 \rightarrow \cdots \xrightarrow{p_{n-1}\ \ q_{n-1}} A_n$, each actor $A_i \in \{A_2, \ldots, A_{n-1}\}$ is delayed in the combined schedule by $Y_{A_i} \in \mathbb{R}_{\geq 0}$ such that $\sum Y_{A_i} \leq s_2 - s_1$. In order that A_1 does not wait for the extra delay Y_{A_2} of A_2, the size of the first buffer could be increased by $\zeta_1 = \left\lceil \frac{Y_{A_2}}{t_{A_1}} \right\rceil p_1$ (*i.e.,* the size is equal to $\theta^u_{A_1,A_2} + \zeta_1$). Similarly, in order that A_2 does not wait for the extra delay Y_{A_3} of A_3, the size of the second buffer could be increased by $\zeta_2 = \left\lceil \frac{Y_{A_3}}{t_{A_2}} \right\rceil p_2$. Finally, in order that A_{n-1} does not wait for the extra delay $(s_1 - s_1) - \sum Y_{A_i}$, the size of the last buffer could be increased by $\zeta_{n-1} = \left\lceil \frac{(s_1 - s_1) - \sum Y_{A_i}}{t_{A_{n-1}}} \right\rceil p_{n-1}$. So, assuming that all tokens have the same size, we need to solve the following optimization program:

$$\textbf{Min} \quad \zeta_1 + \zeta_2 + \ldots + \zeta_{n-1}$$
$$\textbf{Subject to} \quad Y_{A_i} \geq 0 \ \wedge \ \sum Y_{A_i} \leq s_2 - s_1$$

For graph G_c, we have $s_2 - s_1 = 12$, and hence, $\zeta_1 = \lceil \frac{Y_B}{4} \rceil 4$ and $\zeta_2 = \lceil \frac{12-Y_B}{3} \rceil 3$. So, the optimization problem consists in finding $0 \leq Y_B \leq 12$ that minimizes $\lceil \frac{Y_B}{4} \rceil 4 - \lfloor \frac{Y_B}{3} \rfloor 3 + 12$.

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* if each actor imposes a higher load than its successor or if 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* $(\forall i.\ z_{A_i} t_{A_i} \geq z_{A_{i+1}} t_{A_{i+1}}) \vee (\forall i.\ z_{A_i} t_{A_i} \leq z_{A_{i+1}} t_{A_{i+1}})$

Let $\theta_{A_i,A_{i+1}}$ be the size of the buffer between A_i and A_{i+1} as computed in Section 4.2. This size allows the single edge graph $A_i \xrightarrow{p_i \quad q_i} A_{i+1}$ to reach its maximal throughput.

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

Note that Property 6.3 is only a *sufficient* condition simply 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. E.g., let $G_d = A_1 \xrightarrow{2 \quad 2} A_2 \xrightarrow{4 \quad 3} A_3$ such that $t_{A_1} = 28$, $t_{A_2} = 20$ and $t_{A_3} = 15$. We have $\theta_{A_2,A_3} = 12$, but the minimum size of channel $A_2 \to A_3$ is actually 9. Indeed, as illustrated in Fig. 18, even if this size delays firings of A_2 (see the 4th firing), the introduced delay does not prohibit A_1 from firing consecutively.

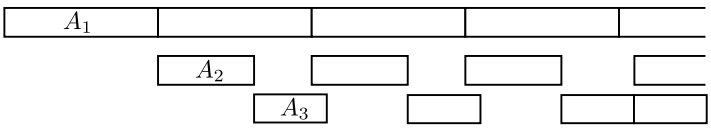


Figure 18: ASAP execution of G_d.

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 $\bigsqcap$. The computed buffer sizes on both sub-chains allow the actors at the "top" to run consecutively.

Unfortunately, Property 6.3 does not hold for an arbitrary chain (*i.e.,* neither ascending, descending nor of the form $\bigsqcap$) (see Example **??** in the appendix). Our solution is to put such an arbitrary chain under the form $\bigsqcap$ by using the same approach as in the proof of Property 6.3; *i.e.,* by increasing the execution times of some actors (without exceeding the maximum load $\mathcal{P}_G$), then computing the buffer sizes as in Property 6.3, and finally restoring the original execution times. Fig. 19 illustrates this solution.

Any chain on the form $\bigsqcap$ obtained by increasing the load of the actors of the original chain is a valid solution. For example, the chain $A \to B \to \ldots \to K$ of Fig. 19 can be transformed into the red chain which is of the form $\bigsqcap$. 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. In the experimental section, we will compare 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 $\theta^u_{A,B} = 2(p + q - \gcd(p, q))$.

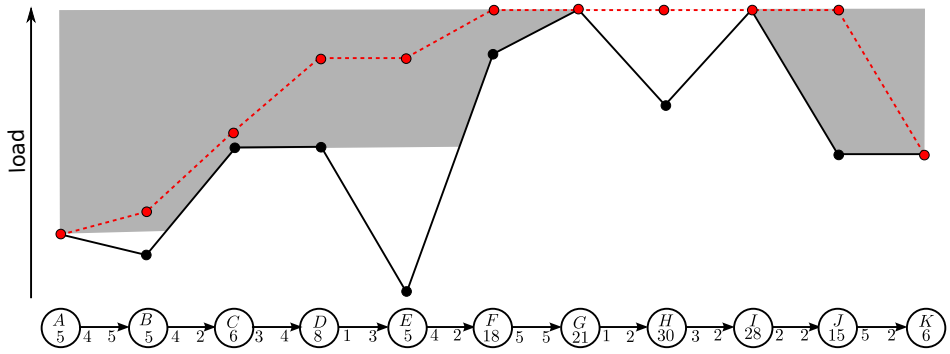


Figure 19: Transformation of a chain to a $\sqcap$ form.

• *MIN:* All loads are raised up to the bottom boundary of the gray area. In Fig. 19, 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 the following criteria. Let $P_A = z_A t_A$ denote the load of actor A.

- According to Fig. 9, the computation of the minimum buffer size $\theta_{A,B}$ of graph $A \xrightarrow{p \quad q} B$, where A imposes a higher load than B, depends on the execution times of A and B. It takes its minimum value $\theta^{\ell}_{A,B} = p + q - \gcd(p,q) + \left\lceil \frac{t_B}{t_A} \right\rceil p$ when the execution times satisfies case (I), and the worst case $\theta^{u}_{A,B}$ when both actors impose the same load. It follows that the difference between the loads of A and B that allows reaching the minimum value of $\theta^{\ell}_{A,B}$ should be

$$\frac{P_A}{P_B} \geq \frac{q}{p} \left\lceil \frac{p}{q} \right\rceil \tag{29}$$

  For instance, for the sub-chain $A \to B \to \ldots \to G$ in Fig. 19, we have $P_A = 50$ and $P_G = 168$. The new distribution of the loads should be a monotone distribution in the gray area and hence in the interval $[50, 168]$. In order to achieve minimum buffer sizes, the new loads of each actor and its predecessor (using duality) have to satisfy Eq.(29). However, this is not possible in all cases since the interval $[P_A, P_G]$ could be very small. The next observation handles this problem.

- The gain on buffer size that comes from satisfying Eq.(29) is at most equal to $\omega_{A,B} = \theta^{u}_{A,B} - \theta^{\ell}_{A,B}$. So,

$$\omega_{A,B} = q - \gcd(p,q)$$

  This expected size gain is used to prioritize the treatment of channels.

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.

6.3 Exact numerical analysis

The SDF^3 tool [23] proposes an exact algorithm [22] that starts by sizing each channel $A \xrightarrow{p\ \ q} B$ in the graph as the lower bound $p+q-\gcd(p,q)$. If this buffer sizes distribution (*i.e.,* assignment of all buffer sizes in the graph) does not allow the desired throughput, a new distribution is created by increasing the size of only one buffer by $\gcd(p,q)$. This algorithm suffers from combinatorial state explosion. In our experimental analysis, we use a different exact algorithm based on a dichotomic search (Algorithm 2). It takes as an input a SDF graph G and an upper bound buffer sizes distribution S^u that allows a maximal throughput, and which can be obtained by the algorithms described in the previous section.

ALGORITHM 2: Exact algorithm for minimum buffer sizes problem

Input: SDF graph G, upper bound solution S^u.
Output: Minimum buffer sizes distribution S^b.
$S^b = S^u$;
$\mathcal{B} = \{B(S) | B(S^\ell) < B(S) \leq B(S^u)\}$ in ascending order;
$mingcd = \min\limits_{A \xrightarrow{p\ \ q} B \in E} (\gcd(p,q))$;
$begin = 1$;
$end = |\mathcal{B}|$;
while $begin < end - 1$ **do**
  $middle = \lfloor (begin + end)/2 \rfloor$;
  $k = middle$;
  $found = false$;
  while *True* **do**
    $Dist = \{S | B(S) = \mathcal{B}[k]\}$;
    if $\exists S \in Dist, \mathcal{T}_G(S)$ *is maximal* **then**
      $found = true$;
      $S^b = S$;
      break;
    else if $k-1 > begin \wedge \mathcal{B}[k-1] + mingcd > \mathcal{B}[middle]$ **then** k-- ;
    else break ;
  if *found* **then** $end = k$;
  else $begin = middle$;
return S^b;

Let $B(S)$ denote the sum of buffer sizes in a distribution S. Algorithm 2 looks for the best distribution S^b with minimal $B(S^b)$ that allows the maximal throughput. We know that $B(S^\ell) < B(S^b) \leq B(S^u)$ such that S^ℓ is a lower bound buffer sizes distribution where the size of each channel $A \xrightarrow{p\ \ q} B$ is equal to $p+q-\gcd(p,q)$. We also know that, for any minimal buffer sizes distribution, the size of each channel $A \xrightarrow{p\ \ q} B$ should be a multiple of $\gcd(p,q)$. Thus, Algorithm 2 checks only distributions that satisfy these two conditions.

Let $mingcd$=$\min\{\gcd(p,q) | A \xrightarrow{p\ \ q} B \in E\}$. For a given buffering requirement k, if every distribution S with $B(S)$=k does not allow the maximal throughput, then it may be possible that there exists a *smaller* distribution S_0 (*i.e.,* $B(S_0)$<k) that allows such throughput. However, we must have $B(S_0) + mingcd > k$. Otherwise, for instance, if $B(S_0) + mingcd = k$, then there exists a distribution S_1 with $B(S_1) = k$ such that S_1 dominates S_0 (*i.e.,* $\forall e \in E, S_1(e) \geq S_0(e)$). But, since S_1 does not allow a maximal throughput, neither should S_0.

The algorithm uses a dichotomic search that first checks whether there is a buffer sizes

distribution, whose total size is at the middle between the lower bound $B(S^\ell)$ and the upper bound solution $B(S^u)$ (*i.e.,* total size equals $\mathcal{B}[middle]$), which allows the graph to achieve its maximal throughput. If there is no such distribution, we check every distribution S such $B(S) + mingcd > \mathcal{B}[middle]$. If there is no distribution ($found = false$), the algorithm proceeds to the top of the search space ($begin = middle$), otherwise it proceeds to the bottom one ($end = k$).

Example 6.1. *Consider graph* $G = A \xrightarrow{8 \quad 14} B \xrightarrow{15 \quad 9} C$ *with* $t_A = 7$, $t_B = 12$ *and* $t_C = 13$. *The repetition vector of this graph is* $\vec{z} = [21, 12, 20]$, *and the minimal period is* $\mathcal{P}_G = z_C t_C = 260$. *The upper bound buffer sizes distribution is* $S^u = [\theta^u_{A,B} = 40, \theta^u_{B,C} = 42]$ *with* $B(S^u) = 82$. *The lower bound distribution is* $S^\ell = [20, 21]$ *with* $B(S^\ell) = 41$.

Algorithm 2, illustrated in Fig. 20, starts by checking distributions at the middle between $B(S^u)$ *and* $B(S^\ell)$, i.e., *distributions such that* $B(S) = 56$ *(solid blue line in Fig. 20). There are three possible distributions:* $[20, 36]$, $[26, 30]$ *and* $[32, 40]$. *Neither of them allows to achieve the maximal throughput. Distribution with* $B(S) < 56$ *and* $B(S) + mingcd > 56$, i.e., *distributions* $[22, 33]$, $[28, 27]$ *and* $[34, 31]$ *(dashed blue line), are not dominated by the already checked distributions, and need to be checked. Neither of them allows to achieve the maximal throughput. This implies that all distributions in the bottom part (dominated by the checked distributions) will not allow a maximal throughput. Therefore, the dichotomic search moves to the upper part.*

At the middle of the upper part (green line), there is a distribution $S = [26, 36]$ *(red dot the green line) that allows a maximal throughput. The algorithm hence proceeds to the bottom part. The process continues: magenta lines (miss), brown line (hit at* $[26, 33]$, *red dot). The algorithm stops and returns* $S^b = [26, 33]$.

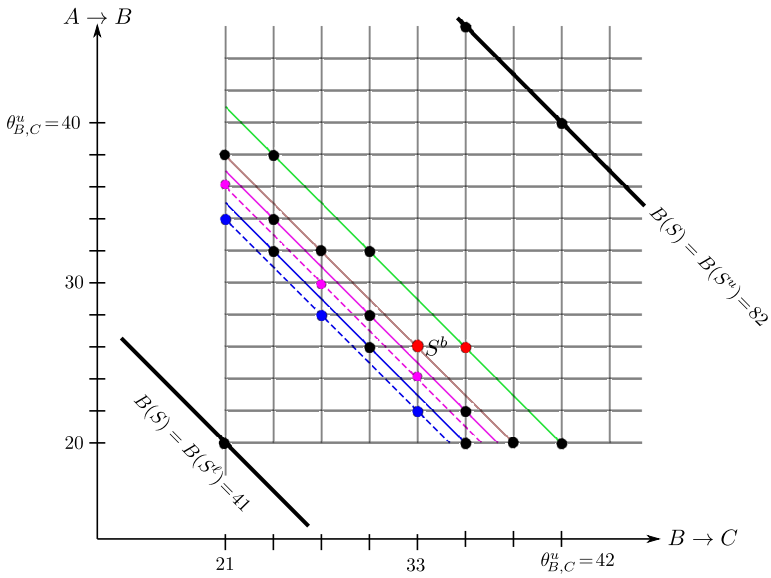


Figure 20: Illustration of Algorithm 2.

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 $\mathcal{L}_G(n)$. 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.*). In this paper, the computed upper bound is used to under-approximate the maximal achievable throughput.

An acyclic SDF graph 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 with the same repetition vector as G, we have the following property.

Property 7.1. *For any acyclic SDF graph G,*

$$\forall i.\ \mathcal{L}_G(i) = \max_{g \in \mathcal{G}(G)} \{\mathcal{L}_g(i)\}$$

According to Property 7.1, the problem reduces to computing the multi-iteration latency of each chain in the graph. Each chain spreads from a source actor to a sink actor.

For each chain $A \xrightarrow{p_1 \quad q_1} B \xrightarrow{p_2 \quad q_2} C \rightarrow \cdots \rightarrow Z$, we compute an upper bound of its multi-iteration latency for n iterations, denoted by $\hat{\mathcal{L}}_{A \rightarrow Z}$ (we omit n for the sake of conciseness). We can compute exactly $\mathcal{L}_{A \rightarrow B}$ as described in Section 4.3. However, since the technique assumes that the producer can run consecutively, 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. 21, the exact multi-iteration latency of chain $A \rightarrow B \rightarrow C$ is $\mathcal{L}_G(n) = \hat{\mathcal{L}}_{A \rightarrow B} + \delta$ where δ is the remaining execution time of C after the end of B.

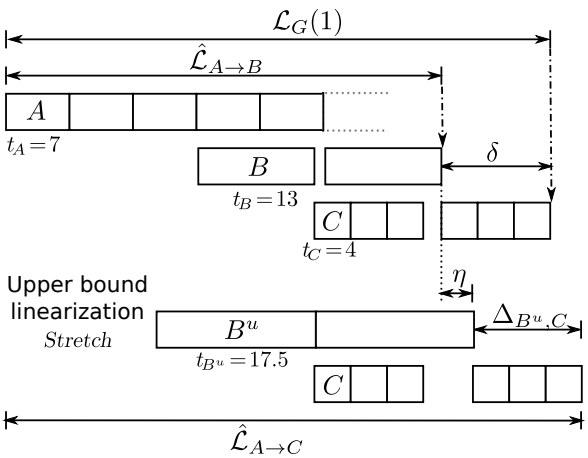


Figure 21: Computation of $\hat{\mathcal{L}}_{A \rightarrow C}$ of a chain $A \xrightarrow{2 \quad 5} B \xrightarrow{3 \quad 1} C$.

If $\eta = f_{B^u}(nz_B) - \hat{\mathcal{L}}_{A \rightarrow B}$ (*i.e.,* the approximation introduced by the upper bound linearization), then $\delta \leq \Delta_{B^u,C} + \eta$. Indeed, $\Delta_{B^u,C}$ (computed by Equations of Fig. 11) 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.
In case of *Push* (Fig. 22), 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 case of *Stretch* (Fig. 21), 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

$$\hat{\mathcal{L}}_{A\to C} \leq \min\{\hat{\mathcal{L}}_{A\to B} + \Delta_{B,C}, f_{B^s}(nz_B) + \Delta_{B^s,C}\} \tag{30}$$

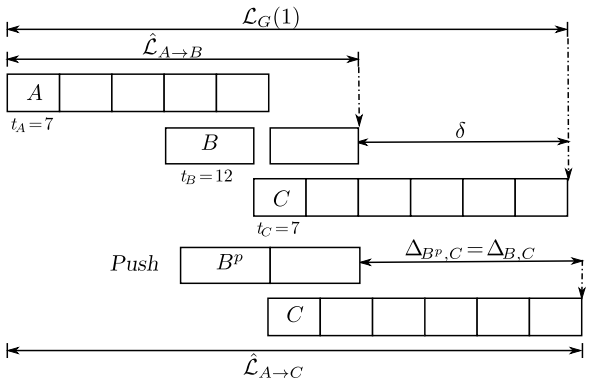


Figure 22: Computation of $\hat{\mathcal{L}}_{A\to C}$ of a chain $A \xrightarrow{2\ \ 5} B \xrightarrow{3\ \ 1} C$ using *Push* method.

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 $A \to B \to C$.

Example 7.1. Let us consider the graph $G = A \xrightarrow{p_1\ \ 1} B \xrightarrow{p_2\ \ 1} C$ which has five parameters: the production rates p_1, p_2 and the execution times t_A, t_B and t_C. Suppose that, at a given design stage, the only known information is that $p_i \in [1, 10^2]$ and that execution times are in $[10^3, 10^4]$. In parametric dataflow models, it is possible that parameters change before the execution enters the steady state. Therefore, a safe over-approximation of the minimal period $\mathcal{P}_G$ of the graph is $\hat{\mathcal{L}}_G(1)$. For this example, we want to check the following property $P_m : \ell_G(1) \leq m\mathcal{P}_G$ where $m \in \mathbb{N}^+$; *i.e.,* the multi-iteration latency of the first iteration does not over-approximate the minimal period by more than m times. A naive approach would be to check this property for all configurations of rates and execution times parameters *i.e.,* for more than 7×10^{15} configurations. With our approach, we can solve this problem analytically. First, we generate a symbolic program

Prog (*i.e.,* a set of equations) that computes $\hat{\mathcal{L}}_G(1)$. This step is polynomial in the number of channels of the graph and depends neither on the parameters nor on the number of iterations. Then, we check if $Prog \wedge P_m$ is satisfied for all configurations. The repetition vector of this graph is $\vec{z} = [1, p_1, p_1p_2]$. The generated program *Prog*, after simplification of the equations, is

$$\hat{\mathcal{L}}_G(1) = \begin{cases} t_A + p_1t_B + p_2t_C & \text{if } t_B \geq p_2t_C \\ t_A + t_B + p_1p_2t_C & \text{otherwise} \end{cases}$$

Since $\mathcal{P}_G = \max(t_A, p_1t_B, p_1p_2t_C)$, it is easy to check for all three possible maximal values (t_A, p_1t_B or $p_1p_2t_C$) that P_m holds for $m \geq 3$ but not for $m < 3$.

Note 7.1. Instead of analyzing separately all the chains of a DAG, 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 $\hat{\mathcal{L}}_{A\to C\to D}$.

Note 7.2. According to the duality theorem, the multi-iteration latencies of a chain $A \to \cdots \to Z$ and 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.

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 nth 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 nth 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

$$\ell_G(n) = \mathcal{L}_G(n) - (n-1)z_At_A \tag{31}$$

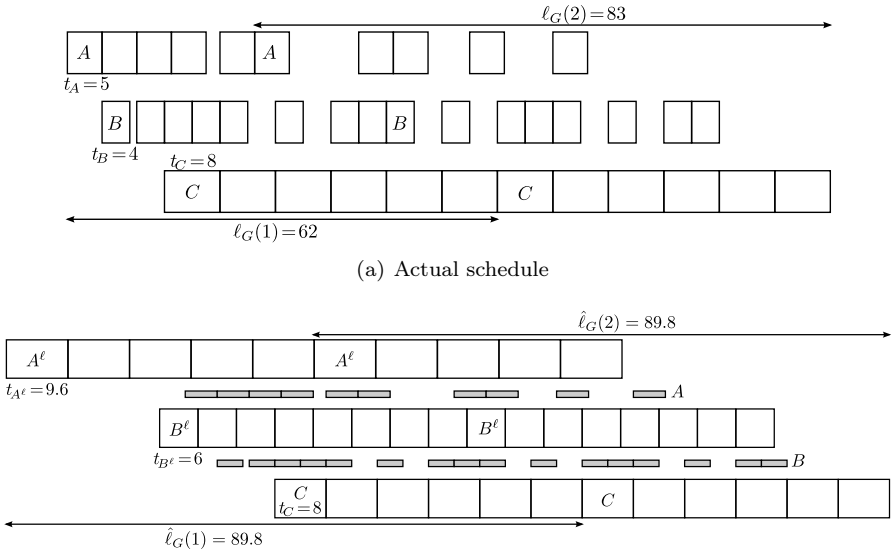
However, if the source actor does not impose the highest load, then $\ell_G(n)$ as given by Eq. (31) is unbounded unless the channels are bounded. Therefore, as in the case of the graph $A \xrightarrow{p\ \ q} B$, we need to consider the buffer sizes when computing the input-output latency.

Consider for instance the chain $A \xrightarrow{8\ \ 5} B \xrightarrow{3\ \ 4} C$ 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. 23(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. (31) to compute the input-output latency.

As illustrated in Fig. 23(b), our solution consists in using a lower bound backward linearization (Section 5.2). Starting by the end of the chain (*i.e.,* 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^s are advanced compared to 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 backward linearize A. 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. (31)) 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 (83).



(a) Actual schedule

(b) Linearized schedule

Figure 23: Input-output latency computation.

The previous approach can be applied to any arbitrary chain where the sink actor imposes the highest load. If the actor with highest load[1], denoted H, is in the middle of a chain $A \rightarrow \ldots \rightarrow H \rightarrow \ldots \rightarrow Z$, 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.

8 Experiments

8.1 Buffer sizing

In this section, we compare 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 (Algorithm 2) using many randomly generated SDF graphs and some real benchmarks.

First, we compare[2] the results of *MIN* with those of *MAX* (*i.e.,* safe upper bounds $2(p + q - \gcd(p, q))$) using two million randomly generated chains of 10 actors where the production and consumption rates (resp. execution times) are uniformly distributed over the interval $[1, 20]$ (resp. $[1, 200]$). The number of firings per iteration, $\sum_X z_X$, of every generated chain has been bounded by 6×10^3. For each graph, we compute the ratio of the sum of the buffers sizes obtained by algorithm *MAX* to those obtained by algorithm *MIN*. Each black dot in Fig.24 represents the obtained ratio for one graph, while the red line represents the average of ratios. Fig. 24 shows that, in average, algorithm *MIN* reduces the total buffer sizes by 8% compared to the upper bounds.

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

[2]Without lost of generality, tokens are assumed to have the same size.

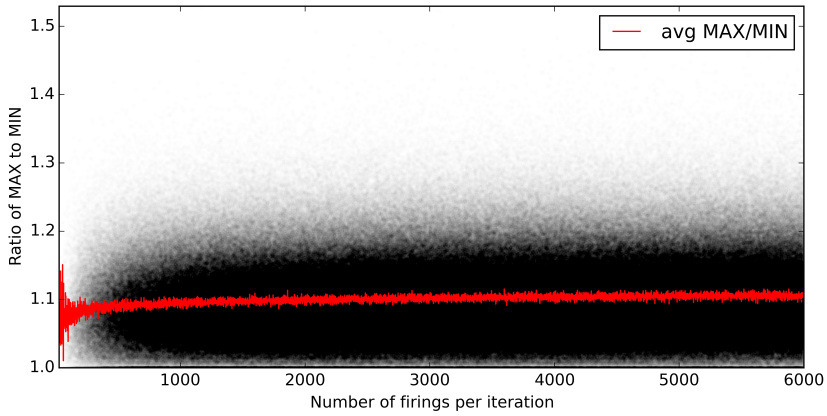


Figure 24: *MAX* vs. *MIN*.

Fig.25 shows a comparison between the *OPT* and *MIN* algorithms with the same settings as for Fig. 24. For instance, the blue curve (*avg MAX/OPT*) represents the average ratio of the results obtained by *MAX* to those of *OPT*. This figure shows that, in 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*.

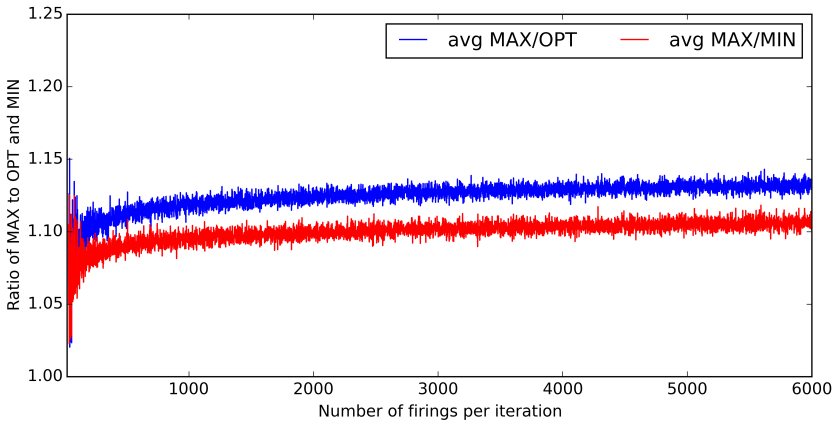


Figure 25: *MAX* vs. *MIN*.

We also compare the results of the heuristic with the exact minimum buffer sizes computed by Algorithm 2 (which we denote by *EXACT*). Due to the exponential complexity of the minimum buffer sizes problem, we evaluate our approach on only 10^4 randomly generated chains of 4 actors where the production and consumption rates (resp. execution times) are uniformly distributed over the interval $[1, 10]$ (resp. $[1, 100]$). The blue (resp. red) line in Fig. 25 represents the average of the ratios of the exact (*EXACT*) (resp. approximate (*OPT*)) solution to the

upper bounds (*MAX*). In average, the *OPT* heuristic over-approximates the exact solution by 25%. Furthermore, Fig. 25 also shows that *MAX* over-approximates *EXACT* by 55% in average. Hence, by extrapolating this result, our *OPT* heuristic would over-approximate the exact solutions of Fig. 24 by 37% in average.

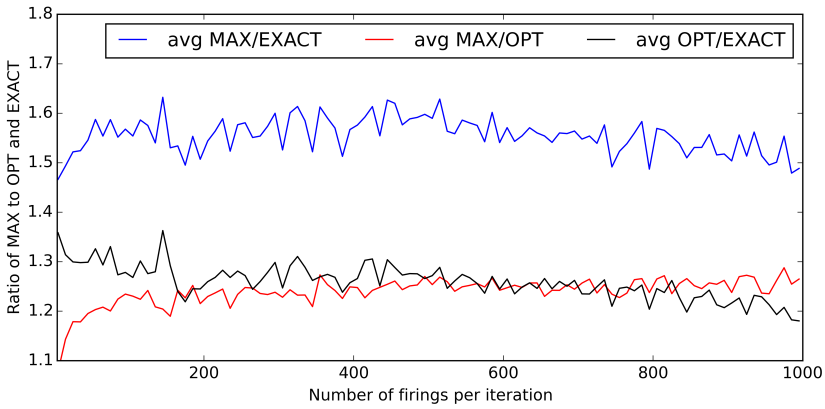
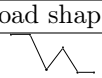
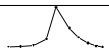
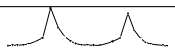


Figure 26: *OPT* vs. exact solution.

Finally, we evaluate the heuristic using five real applications: the H.263 decoder, the data modem and sample rate converter from the SDF^3 benchmarks [23], the fast fourier transformer (FFT), and the time delay equalizer (TDE) from the StreamIt benchmarks [24]. All these graphs have a chain structure. Table 1 shows some characteristics of these applications together with the obtained results. Our approach improves better the upper bounds in case of chains with a $\sqcap$ 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 1: Experimental results for real benchmarks.

graph	# actors	$\sum_A z_A$	load shape	Upper bound	Optimal size	Heuristic
modem	6	37		32	20	30
sample con.	6	612		60	34	57
H.263 dec.	4	1190		2378	1257	1257
FFT	11	94		992	504	808
TDE	27	2867		7328	3680	5272

8.2 Latency computation

We first evaluate our approach for computing the multi-iteration latency using randomly generated chains of 10 actors. Five million chains have been generated such that the production and consumption rates are uniformly distributed over the interval $[1, 10]$ while the execution times are

uniformly distributed over then interval $[1, 100]$. The number of firings per iteration (*i.e.,* $\sum_A z_A$) of every generated chain has been bounded by 2×10^3. The exact multi-iteration latency of the first iteration (*i.e.,* $\mathcal{L}_G(1)$) is compared with the latency computed using our approach (denoted by $\hat{\mathcal{L}}_{A_1 \to A_{10}}$); and we report the ratio $\frac{\hat{\mathcal{L}}_{A_1 \to A_{10}}}{\mathcal{L}_G(1)}$ (*i.e.,* the over-approximation). Each black dot in the upper part of Fig. 27 represents the ratio for one graph, while the red line represents the average of ratios. The average over-approximation is negligible when the number of firings per iteration 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%) 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 tends 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 bottom part of Fig. 27 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.

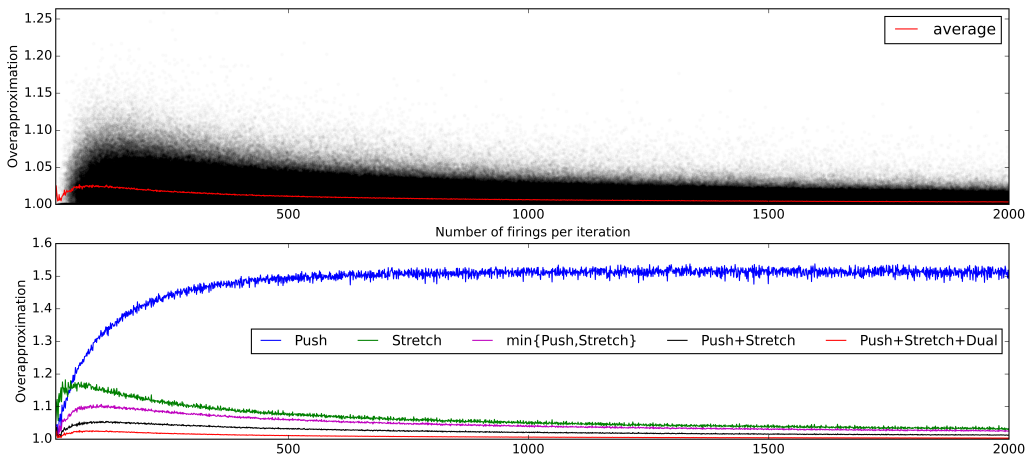


Figure 27: Multi-iteration Latency computation.

Table 2 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.

Finally, we evaluate our approach for computing the input-output latency using 10^5 randomly generated chains of 9 actors. Chains are generated such that the production and consumption rates are in the interval $[1, 10]$ while the execution times are in the interval $[1, 100]$. The number of firings per iteration of every generated chain has been bounded by 2×10^3. The last actor in each chain imposes the highest load, and the size of each channel $A \xrightarrow{p \quad q} B$ is equal to

Table 2: 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

$2(p + q - \gcd(p, q))$. Each black dot in Fig. 28 represents the obtained ratio $\frac{\hat{\ell}_G}{\ell_G}$ for one graph, while the red line represents the average of ratios. Note that the exact computation of ℓ_G is very time-consuming since it requires a simulation of the schedule until reaching the steady state (hundreds of iterations for some chains). Therefore, we have limited our experiment to only 10^5 chains of 9 actors. Fig. 28 shows that our analysis over-approximates the exact computation, in average, by at most 13%. The over-approximation is less noticeable for graphs with large input-output latencies.

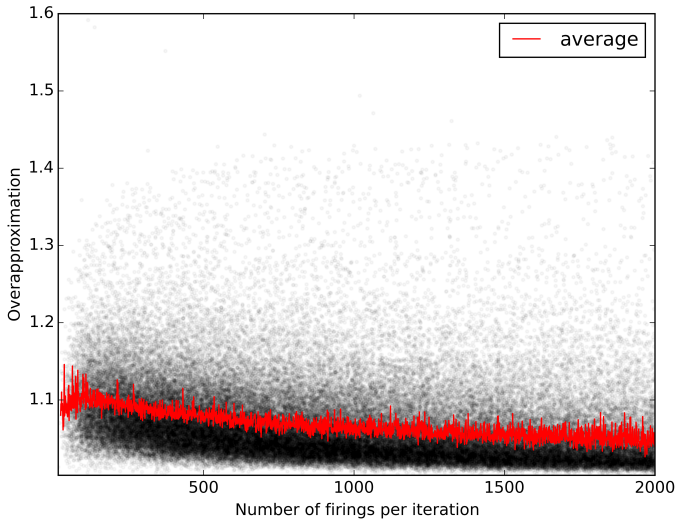


Figure 28: Input-output latency computation.

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.,* [2, 9]).

There is no exact analytical solution to check the liveness of a graph with buffers with fixed bounds. In [2] and [3], 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 [1], the minimum buffer size for which the simple graph $A \xrightarrow{p\ \ q} B$ is live is equal to $p + q - \gcd(p, q)$[3]. This however does not imply that any graph whose channels are sized this way is live. Still, this analytical equation is used in many algorithms of buffer sizing to compute a lower bound on buffer sizes as a starting solution ([3, 22]).

Let $\vec{s}_i$ denotes the *token timestamp vector*, where each entry corresponds to the production time of tokens in the ith iteration of the graph. Then, as shown in [10], the max-plus algebra can be used to express the evolution of the token timestamp vector: $\vec{s}_i = M\vec{s}_{i-1}$. It has been proved that 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 [18, 19]. However, this works only in cases where Eq. (4) can be somehow simplified to get rid of the ceiling function (*e.g.,* when $p = 1$).

[11] 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. This approach has been extended in [6] to the case of 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.,* [5]). 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 (it depends on the number of initial tokens). Furthermore, experiments on real-life benchmarks show that these approaches result in huge over-approximations (sometimes 7 times the exact value) [5]. In theory, the over-approximation is not even bounded.

10 Conclusion

We have studied analytically the different cases of the execution of a completely parametric single edge dataflow graph $A \xrightarrow{p\ \ q} B$. Enabling patterns are introduced to better characterize the data-dependency between the producer and the consumer. Then, we have 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.

Using these results and forward linearization techniques, we have provided safe upper bounds of buffer sizes of acyclic graphs for maximal throughput. Furthermore, we have proposed a heuristic to improve these bounds for graphs with a chain or a tree structure. Experimental results show that our heuristic improves the 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. The exact solutions was computed using our numerical analysis which is based on a dichotomic search.

We also used 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.

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

Experimental results show that 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 analysis to deal with general (*i.e.,* possibly cyclic) dataflow graphs.

References

[1] S. S. Battacharyya, E. A. Lee, and P. K. Murthy. *Software synthesis from dataflow graphs.* Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[2] V. Bebelis, P. Fradet, A. Girault, and B. Lavigueur. BPDF: a statically analyzable dataflow model with integer and boolean parameters. In *Proceedings of the 11th ACM International Conference on Embedded Software*, pages 3:1–3:10, 2013.

[3] E. Bempelis. *Boolean Parametric Data Flow: Modeling - Analysis - Implementation.* PhD thesis, Université Grenoble Alpes, 2015.

[4] B. Bhattacharya and S. S. Bhattacharyya. Parameterized dataflow modeling of DSP systems. *Trans. Sig. Proc.*, 49(10):2408–2421, 2001.

[5] B. Bodin, A. Munier-Kordon, and B. de Dinechin. Periodic schedules for cyclo-static dataflow. In *Proceedings of the 11th Symposium on Embedded Systems for Real-time Multimedia*, pages 105–114, 2013.

[6] M. Damavandpeyma, S. Stuijk, M. Geilen, T. Basten, and H. Corporaal. Parametric throughput analysis of scenario-aware dataflow graphs. In *Proceedings of the 30th International Conference on Computer Design*, pages 219–226, 2012.

[7] J. B. Dennis. First version of a data flow procedure language. In *Programming Symposium, Proceedings Colloque sur la programmation*, pages 362–376, 1974.

[8] K. Desnos, M. Pelcat, J. Nezan, S. S. Bhattacharyya, and S. Aridhi. PiMM: parametrized and interfaced dataflow meta-model for MPSoCs runtime reconfiguration . In *Proceedings of the 2013 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 41–48, 2013.

[9] P. Fradet, A. Girault, and P. Poplavko. SPDF: a schedulable parametric data-flow MoC. In *Design, Automation and Test in Europe Conference and Exhibition*, pages 769–774, 2012.

[10] M. Geilen. Synchronous dataflow scenarios. *ACM Trans. Embed. Comput. Syst.*, 10(2):16:1–16:31, 2011.

[11] A. H. Ghamarian, M. C. W. Geilen, T. Basten, and S. Stuikj. Parametric throughput analysis of synchronous data flow graphs. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 116–121, 2008.

[12] A. H. Ghamarian, S. Stuijk, T. Basten, M. C. W. Geilen, and B. D. Theelen. Latency minimization for synchronous data flow graphs. In *Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools*, pages 189–196, 2007.

[13] G. Kahn. The semantics of a simple language for parallel programming. In *IFIP Congress*, pages 471–475, 1974.

[14] A. Kumar, H. Corporaal, B. Mesman, and Y. Ha. *Multimedia Multiprocessor Systems: Analysis, Design and Management.* Springer Science+Business Media B.V., New York, NY, USA, 2011.

[15] E. A. Lee and D. G. Messerschmitt. Synchronous data flow. In *Proceedings of the IEEE*, pages 1235–1245, 1987.

[16] O. Moreira, T. Basten, M. Geilen, and S. Stuijk. Buffer sizing for rate-optimal single-rate data-flow scheduling revisited. pages 188–201, 2010.

[17] E. Nogues, R. Berrada, M. Pelcat, D. Menard, and E. Raffin. A DVFS based HEVC decoder for energy-efficient software implementation on embedded processors. In *2015 IEEE International Conference on Multimedia and Expo*, pages 1–6, 2015.

[18] M. Skelin, M. Geilen, F. Catthoor, and S. Hendseth. Worst-cas throughput analysis for parametric rate and parametric actor execution time scenario-aware dataflow graphs. In *Proceedings of the 1st International Workshop on Synthesis of Continuous Parameters*, pages 65–79, 2014.

[19] M. Skelin, M. Geilen, F. Catthoor, and S. Hendseth. Parametrized dataflow scenarios. In *Proceedings of the 12th International Conference on Embedded Software*, pages 95–104, 2015.

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

[21] S. Stuijk, T. Basten, M. Geilen, H. Corporaal, and M. Damavandpeyma. Throughput-constrained DVFS for scenario-aware dataflow graphs. In *Proceedings of the 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium*, pages 175–184, 2013.

[22] S. Stuijk, M. Geilen, and T. Basten. Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs. In *Proceedings of the 43rd Annual Design Automation Conference*, pages 899–904, 2006.

[23] S. Stuijk, M. Geilen, and T. Basten. SDF^3: SDF for free. In *Proceedings of the 6th International Conference on Application of Concurrency to System Design*, pages 276–278, 2006.

[24] W. Thies and S. Amarasinghe. An empirical characterization of stream programs and its implications for languages and compiler design. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, pages 365–376, 2010.

A Throughput and duality

Proof. (**Property 3.1**) This is easily shown by considering the corresponding HSDF graph. As illustrated in Fig. 29(a), the SDF-to-HSDF transformation algorithm [20] replicates each actor A in the original graph z_A times (recall that $\vec{z}$ is the repetition vector), each instance represents a firing of A in one iteration. Then, each edge $A \xrightarrow{p\ \ q} B$ in the original graph is transformed into $p\, z_A$ edges, each one representing a data dependency between a firing of A and a firing of B. Let $\mathit{HSDF}(G)$ denote the HSDF graph equivalent to the acyclic graph G. The only cycles that appear in $\mathit{HSDF}(G)$ are the results of the transformation of self-edges used to disable auto-concurrency.

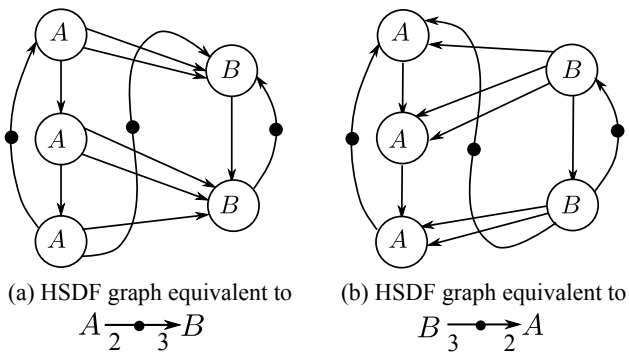


Figure 29: SDF-to-HSDF transformation of a graph and its dual.

For each actor A, its corresponding cycle contains one delay and z_A instances of A. Thus, the cycle mean is equal to $z_A t_A$, and the MCM is hence equal to $\max_{A \in V}\{z_A t_A\}$. □

Proof. (**Theorem 3.1**) We first prove that the graph $\mathit{HSDF}(G)$ is the dual of $\mathit{HSDF}(G^{-1})$. As described in the previous proof, the SDF-to-HSDF transformation is compositional in the sense that each channel is transformed independently of the rest of the graph. Hence, it is sufficient to prove that $\mathit{HSDF}(A \xrightarrow{p\ \ q} B)$ and $\mathit{HSDF}(B \xrightarrow{q\ \ p} A)$ are dual to each other as illustrated in Fig. 29 (see Property A.1). It follows that each cycle in $\mathit{HSDF}(G)$ has a dual in $\mathit{HSDF}(G^{-1})$. Therefore, both graphs have the same MCM and the same throughput *i.e.,* $\mathcal{T}_G = \mathcal{T}_{G^{-1}}$.

The second step in this proof is to show that $\forall i.\ \mathcal{L}_{\mathit{HSDF}(G)}(i) = \mathcal{L}_{\mathit{HSDF}(G^{-1})}(i)$. The multi-iteration latency of the first i iterations of an HSDF graph can be obtained by searching the longest path in a directed acyclic graph (DAG) obtained by unfolding the HSDF graph for i iterations and deleting any edge with an initial token (since this edge does not impose a precedence constraint between firings).

Thus, by unfolding $\mathit{HSDF}(G)$ and $\mathit{HSDF}(G^{-1})$ for i iterations, we obtain two DAGs that are dual to each other since the unfolding transformation preserves duality. 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 hence $\mathcal{L}_{\mathit{HSDF}(G)}(i) = \mathcal{L}_{\mathit{HSDF}(G^{-1})}(i)$. □

Property A.1. *Let $G = A \xrightarrow{p\ \ q} B$. Graphs $\mathit{HSDF}(G)$ and $\mathit{HSDF}(G^{-1})$ are dual to each other.*

Proof. (**Property A.1**) The SDF-to-HSDF transformation of $A \xrightarrow{p\ \ q} B$ duplicates actor A z_A times and actor B z_B times. Furthermore, it creates pz_A (or equivalently qz_B) dependencies between instances of A and instances of B. These dependencies are named as illustrated in Fig. 30.

The ith token produced by A in G creates a dependency $A^j \longrightarrow B^k$ in the graph $\texttt{HSDF}(G)$ with $j = (i-1) \bmod pz_A$ and $k = (i+d-1) \bmod pz_A$ where d is the number of initial tokens in channel $A \rightarrow B$. We then prove that $\exists i'$ such that the i'^{th} token produced in graph G^{-1} creates a dependency $B^{pz_A-1-k} \longrightarrow A^{pz_A-1-j}$ in the graph $HSDF(G^{-1})$. For instance, in Fig.30.(a) we have the dependency $A^5 \rightarrow B^0$ in $\texttt{HSDF}(G)$. Its counterpart in Fig. 30.(b) is the data-dependency $B^{6-1-0=5} \rightarrow A^{6-1-5=0}$.

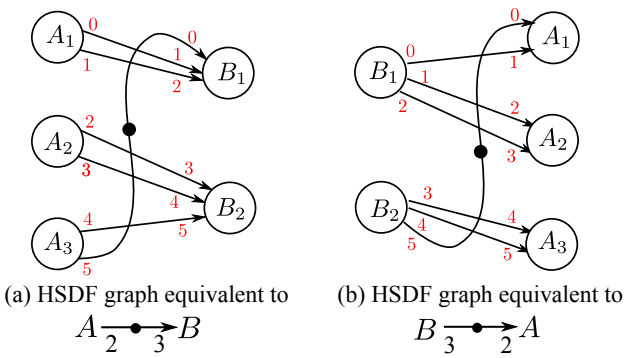


Figure 30: Duality of $HSDF(A \xrightarrow{p\ \ q} B)$ and $HSDF(B \xrightarrow{q\ \ p} A)$.

Therefore, we need to prove that

$$\forall i \exists i'.\ \begin{cases} (i'-1) \bmod pz_A = pz_A - k - 1 \\ (i'+d-1) \bmod pz_A = pz_A - j - 1 \end{cases}$$

Hence, by simplification, $\forall i \exists i'.\ (i+i'+d-2) \bmod pz_A = pz_A - 1$. This equation is satisfiable independently of d, p and z_A. This process goes both ways; *i.e.,* there is a bijection between edges in $HSDF(G)$ and edges in $HSDF(G^{-1})$. Therefore, if we rename actors in $HSDF(G^{-1})$ in the reverse order, then each edge in $HSDF(G)$ has a reversed counterpart in $HSDF(G^{-1})$. Note that renaming actors (*i.e.,* A_{z_A+1-i} instead of A_i) does not alter the ASAP execution since instances of an actor has the same execution time.

Now, we prove that the numbers of initial tokens in edges $A^j \longrightarrow B^k$ and $B^{pz_A-1-k} \longrightarrow A^{pz_A-1-j}$ are the same. The number of initial tokens in $A^j \longrightarrow B^k$ is equal to $\left\lfloor \frac{d}{pz_A} \right\rfloor$ if $d \bmod pz_A \leq k$; and $\left\lfloor \frac{d}{pz_A} \right\rfloor + 1$ otherwise. While the number of initial tokens in edge $B^{pz_A-1-k} \longrightarrow A^{pz_A-1-j}$ is equal to $\left\lfloor \frac{d}{pz_A} \right\rfloor$ if $d \bmod pz_A \leq (pz_A - 1 - j)$; and $\left\lfloor \frac{d}{pz_A} \right\rfloor + 1$ otherwise. But since $(d \bmod pz_A \leq k) \Leftrightarrow (d \bmod pz_A \leq (pz_A - 1 - j))$, the property is satisfied. □

B The parametric graph $A \xrightarrow{p\ \ q} B$

B.1 Enabling patterns

Proof. (**Property 4.1**)

• Case (A.1): ($p = kq$) The repetition vector is $\vec{z} = [1, k]$. Each firing of A enables k firings of B and the enabling pattern is $A \rightsquigarrow B^k$.

• Case (A.2): ($p = kq + r$ and $q \leq 2r$) The first firing of A enables only k firings of B, yielding the pattern $A \rightsquigarrow B^k$. The number of remaining tokens in the channel after sequence AB^k is equal to r. Since $2r \geq q$, the second firing of A enables $k+1$ firings of B, yielding the aggregated pattern $A \rightsquigarrow B^k; A \rightsquigarrow B^{k+1}$. It follows that pattern $A \rightsquigarrow B^{k+1}$ can be repeated a number of times denoted α_1, after which there will be $r + \alpha_1(r - q)$ tokens left. So, α_1 is the largest integer such that $r + \alpha_1(r - q) \geq 0$. Hence, $\alpha_1 = \lfloor \frac{r}{q-r} \rfloor$.

The next firing of A will only enable k firings of B ($A \rightsquigarrow B^k$). By the same reasoning as above, this will be followed by $[A \rightsquigarrow B^{k+1}]^{\alpha_2}$ where α_2 is the largest integer such that $r + \alpha_1(r - q) + r + \alpha_2(r - q) \geq 0$. Hence, $\alpha_2 = \lfloor \frac{2r}{q-r} \rfloor - \lfloor \frac{r}{q-r} \rfloor$. This process is repeated until all firings of A and B of the iteration take place, yielding the complete pattern

$$\left[A \rightsquigarrow B^k; [A \rightsquigarrow B^{k+1}]^{\alpha_j}\right]^{j=1 \cdot\cdot m}$$

for some m that remains to compute. The number of firings of A in this pattern is equal to $c_A = \sum_{j=1}^{m} 1 + \alpha_j$, while the number of firings of B is equal to $c_B = \sum_{j=1}^{m} k + (k+1)\alpha_j$. Note that since $\sum_{j=1}^{m} \alpha_j = \left\lfloor \frac{m\,r}{q-r} \right\rfloor$, we have $c_A = m + \left\lfloor \frac{m\,r}{q-r} \right\rfloor$ and $c_B = m\,k + (k+1) \left\lfloor \frac{m\,r}{q-r} \right\rfloor$. A correct pattern covers the entire iteration, that is $c_A = z_A$ and $c_B = z_B$. We have $z_A = \frac{q}{\gcd(p,q)}$ and $z_B = \frac{p}{\gcd(p,q)}$, thus $m\,k + (k+1)(\frac{q}{\gcd(p,q)} - m) = \frac{p}{\gcd(p,q)}$ This implies that $m = \frac{q-r}{\gcd(p,q)}$.

• Case (A.3): ($p = kq + r$ and $q > 2r$) Similarly to case (A.2), the first firing of A enables k firings of B. However, since $q > 2r$, the second firing of A enables only k firings. The sequence of numbers of remaining tokens will be

$$0 \xrightarrow{AB^k} r \xrightarrow{AB^k} 2r \xrightarrow{AB^k} \cdots \xrightarrow{AB^k} \beta_1 r \xrightarrow{AB^{k+1}} \beta_1 r + (r - q) \tag{32}$$

So, β_1 is the smallest number for which $\beta_1 r + (r - q)$ is non-negative. Hence, $\beta_1 = \lceil \frac{q-r}{r} \rceil$. At this point, the next firing of A enables $k+1$ instances of B. This process will repeat infinitely with $\beta_j = \lceil \frac{j(q-r)}{r} \rceil - \lceil \frac{(j-1)(q-r)}{r} \rceil$. Sequence (β_j) is periodic with a period of length $\frac{r}{\gcd(p,q)}$. At the end of each period, the number of remaining tokens returns to zero.

• Case (B.1): ($q = kp$) The repetition vector is $\vec{z} = [k, 1]$. Each k firings of A enable one firing of B and the enabling pattern is $A^k \rightsquigarrow B$.

• Case (B.2): ($q = kp + r$ and $p \geq 2r$) $(k+1)$ firings of A are needed to enable the first B. Hence, after $(A^{k+1}B)$, the number of remaining tokens in the buffer is $(p - r)$. If $p \geq 2r$, then only k firings of A are needed to enable the second firing of B; and the number of remaining tokens will be $(p - r) - r$. As in the previous cases, we will have a sequence

$$0 \xrightarrow{A^{k+1}B} (p - r) \xrightarrow{A^k B} (p - r) - r \xrightarrow{A^k B} \cdots \xrightarrow{A^k B} (p - r) - \gamma_1 r \tag{33}$$

So, γ_1 is the largest number for which $(p - r) - \gamma_1 r$ is non-negative; thus $\gamma_1 = \lfloor \frac{p-r}{r} \rfloor$. At this point, $k+1$ firings of A are required to enable another firing of B. This process will repeat infinitely with $\gamma_j = \lfloor \frac{j(p-r)}{r} \rfloor - \lfloor \frac{(j-1)(p-r)}{r} \rfloor$. Note that sequence (γ_j) is periodic with a period of length $\frac{r}{\gcd(p,q)}$. At the end of each period, the number of remaining tokens returns to zero.

• Case (B.3): ($q = kp + r$ and $p < 2r$) Similarly to case (B.2), $(k+1)$ firings of A are needed to enable the first B. However, since $p < 2r$, $(k+1)$ firings of A are required to enable the second firing of B. The sequence of numbers of remaining tokens is then

$$0 \xrightarrow{A^{k+1}B} (p-r) \xrightarrow{A^{k+1}B} 2(p-r) \xrightarrow{A^{k+1}B} \cdots \xrightarrow{A^{k+1}B} \lambda_1(p-r) \xrightarrow{A^{k}B} \tag{34}$$

So, λ_1 is the smallest number for which $\lambda_1(p-r) - r$ is non-negative. Hence, $\lambda_1 = \lceil \frac{r}{p-r} \rceil$. At this point, only k firings of A are required to enable the next firing of B. This process will repeat infinitely with $\lambda_j = \lceil \frac{jr}{p-r} \rceil - \lceil \frac{(j-1)r}{p-r} \rceil$. Sequence (λ_j) is periodic with a period of length $\frac{p-r}{\gcd(p,q)}$. At the end of each period, the number of remaining tokens returns to zero. □

B.2 Minimum buffer size for maximum throughput

Proof. (**Property 4.2**)

The three cases of Fig. 9 should be read as (I) else (II) otherwise (III).

Case (I): *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.*

In terms of the enabling pattern cases identified in Fig. 8, case (I) must be split into two exclusive subcases, (I.1) when $p \geq q$ and (I.2) otherwise. The conjunction of the condition for case (I.1) yields the condition $\text{A.1} \vee ((\text{A.2} \vee \text{A.3}) \wedge (t_A \geq (k+1)t_B))$ where (A.1), (A.2) and (A.3) refer to the exclusive patterns of Fig. 8. The other conditions are obtained similarly.

Case (I), pattern (A.1):

In case $p \geq q$ and $r = 0$, the parallel schedule is $A; [A||B^k]^*$, which means that each k firings of B run in parallel with a firing of A. We have, $\delta_1 = p$, $\delta_2 = 2p$, $\delta_3 = \delta_2 + p - kq = 2p, \ldots$. So, $\theta_{A,B} = \max_i \delta_i = 2p$.

Case (I), pattern (A.2):

Let us prove the result for case (I.1) and pattern (A.2), *i.e.*, $p = kq + r$ with $q \leq 2r$. According to the enabling pattern (A.2), at most $(k+1)$ firings of B can be enabled at a given point. They all run in parallel with one firing of A. Therefore, case (I.1) requires that $t_A \geq (k+1)t_B$.

We first expand the enabling pattern of (A.2) into an infinite pattern in order to compute the minimum buffer size $\theta_{A,B}$ over the infinite execution of the graph:

$$\left[A \rightsquigarrow B^k; \left[A \rightsquigarrow B^{k+1}\right]^{\alpha_j}\right]^{j \in \mathbb{N}^+} \tag{35}$$

For the sake of the proof, since $\alpha_j \geq 1$, we can rewrite Eq. (35) into the equivalent infinite pattern:

$$\underbrace{A \rightsquigarrow B^k; A \rightsquigarrow B^{k+1}}_{\text{prologue}}; \underbrace{\left[[A \rightsquigarrow B^{k+1}]^{\alpha_j - 1}; A \rightsquigarrow B^k; A \rightsquigarrow B^{k+1}\right]}_{\text{block } j}{}^{j \in \mathbb{N}^+} \tag{36}$$

Recall that δ_j denotes the minimum number of tokens in the backward edge such that the jth firing of A can occur immediately after the $(j-1)^{th}$ firing of A. The minimum number of tokens to enable the first firing of A must be $\delta_1 = p$. Then, $\delta_2 = \delta_1 + p = 2p$ because no B has finished before the second A starts. Then, $\delta_3 = \delta_2 + (p - kq) = 2p + r$. Subsequently, $\delta_4 = \delta_3 + (p - (k+1)q) = \delta_3 + (r - q)$, and all α_1 subsequent values of δ_i are obtained from δ_{i-1} by adding $(r - q)$ which is negative. The value of α_j is defined in Fig. 8. Hence, $\delta_{1+(\alpha_1+1)+1} =$

$\delta_3 + \alpha_1(r-q) = 2p + r + \alpha_1(r-q)$. This ends at the last of the α_1 patterns $A \rightsquigarrow B^{k+1}$, so the next value $\delta_{1+(\alpha_1+1)+2}$ is obtained by adding $p - kq = r$ because of pattern $A \rightsquigarrow B^k$, yielding $2p + r + \alpha_1(r-q) + r$.

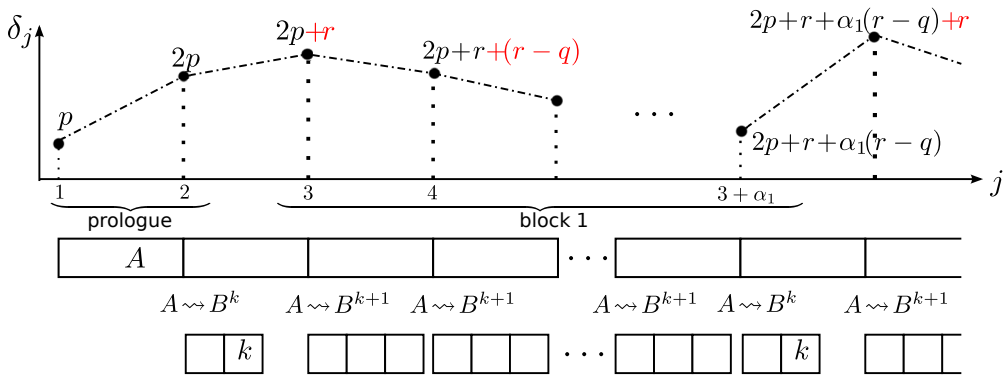


Figure 31: Illustration of sequence (δ_j) in case (I.1) and pattern (A.2).

The computation of the infinite sequence (δ_j) is illustrated in Fig. 31. Within each block j, the subsequence (δ_h) is strictly decreasing because $(r-q)$ is negative, so its maximum value is the value of the entry point, which we denote by $\ell_j = 1 + \sum_{i=1}^{j}(\alpha_i + 1) + 2$. We thus have:

$$\delta_{\ell_j} = 2p + r + \sum_{i=1}^{j}(\alpha_i(r-q) + r) = 2p + r + jr + (r-q)\left\lfloor \frac{jr}{q-r} \right\rfloor$$

It follows that the maximum value of the infinite sequence (δ_j) is:

$$\begin{aligned}
\theta_{A,B} &= \max_{j\in\mathbb{N}} \delta_{\ell_j} \\
&= \max_{j\in\mathbb{N}}(2p + r + jr + (r-q)\left\lfloor \frac{jr}{q-r} \right\rfloor) \\
&= 2p + r + (q-r)\max_{j\in\mathbb{N}}(\frac{jr}{q-r} - \left\lfloor \frac{jr}{q-r} \right\rfloor)
\end{aligned}$$

As a conclusion, $\theta_{A,B} = 2p + r + (q - r - \gcd(p,q)) = 2p + q - \gcd(p,q)$.

Case (I), pattern (A.3):

We write the parallel schedule as $A; [[A||B^k]^{\beta_j}; A||B^{k+1}]^*$. Again, we have $\delta_3 = \delta_2 + p - q = \delta_2 + r, \delta_4 = \delta_3 + p - q = \delta_2 + 2r, \ldots$ (see Fig. 32)

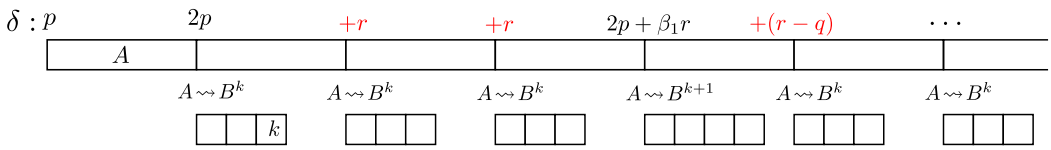


Figure 32: Case (I) pattern (A.3).

Hence, we will have a sequence similar to that of Eq. (32). If S_2 denotes that sequence, then $\theta_{A,B} = \max_i \delta_i = \delta_2 + \max S_2$. Since $(r-q)$ is negative, we have that $\max S_2 = (q-r) +$

$\max_j \left(j(r-q) + r \sum_{i=1}^{j} \beta_i \right) = (q-r) + \max_j \left(r \left\lceil \frac{jq}{r} \right\rceil - jq \right) = (q-r) + r \max_j \left(\left\lceil \frac{jq}{r} \right\rceil - \frac{jq}{r} \right)$. But, $\max_j \left(\left\lceil \frac{jq}{r} \right\rceil - \frac{jq}{r} \right) = \frac{r-\gcd(q,r)}{r}$ [Using Bézout identity]. Thus, $\max S_2 = (q-r) + r - \gcd(q,r)$. Hence, $\theta_{A,B} = 2p + q - \gcd(p,q)$.

Case (I), pattern (B.1):

The first firing of B will not finish before the start of the $(\left\lceil \frac{t_B}{t_A} \right\rceil)^{th}$ firing of A. Let us put $x = k + \left\lceil \frac{t_B}{t_A} \right\rceil$. Hence, $\delta_x = xp$, $\delta_{x+1} = \delta_x + p - q, \ldots, \delta_{x+k} = \delta_x + xp - q = \delta_x, \ldots$ So, $\theta_{A,B} = xp = q + \left\lceil \frac{t_B}{t_A} \right\rceil p$.

Case (I), pattern (B.2):

The first firing of B will not finish before the start of the $(k + 1 + \left\lceil \frac{t_B}{t_A} \right\rceil)^{th}$ firing of A. Let us put $x = k + 1 + \left\lceil \frac{t_B}{t_A} \right\rceil$. Hence, $\delta_x = xp$. We have that $\delta_{x+k} = \delta_x + kp - q = \delta_x - r, \ldots, \delta_{x+\gamma_1 k} = xp - \gamma_1 r, \ldots$ (See Fig. 33)

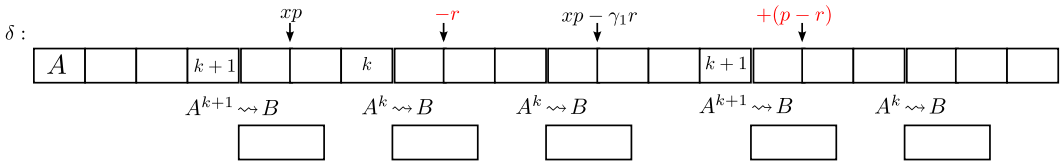


Figure 33: Case (I) pattern (B.2).

This sequence is similar to the sequence in Eq . 33. So, $\theta_{A,B} = \max_i \delta_i = xp + \max_j \left(j(p-r) - r \sum_{i=1}^{j} \gamma_i \right) = xp + r \max_j \left(\frac{jp}{r} - \left\lfloor \frac{jp}{r} \right\rfloor \right) = xp + \max_j (jp \bmod r)$. But, $\max_j (jp \bmod r) = r - \gcd(p,r)$. Therefore, $\theta_{A,B} = p + q - \gcd(p,q) + \left\lceil \frac{t_B}{t_A} \right\rceil p$.

Case (I), pattern (B.3):

Similarly to case (B.2), we have that $\delta_x = xp, \delta_{x+k+1} = \delta_x + (k+1)p - q = \delta_x + (p - r), \ldots, \delta_{x+(k+1)(\gamma_1-1)+k} = \delta_x + (\gamma_1 - 1)(p-r) - r, \ldots$ (See Fig. 34)

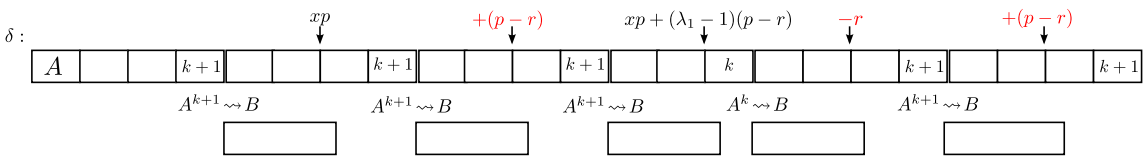


Figure 34: Case (I) pattern (B.3).

This sequence is similar to that of Eq. (34). So, $\theta_{A,B} = xp + (r-p) + \max_j \left(-rj + (p-r) \sum_{i=1}^{j} \lambda_j \right) + r = xp + 2r - p + (p-r) \max_j \left(\left\lceil \frac{jr}{p-r} \right\rceil - \frac{jr}{p-r} \right) = xp + 2r - p + (p - r - \gcd(p,r))$. Thus, $\theta_{A,B} = p + q - \gcd(p,q) + \left\lceil \frac{t_B}{t_A} \right\rceil p$.

Case (II): *Case (I) is not satisfied, but, for any block (*e.g., $[[A \rightsquigarrow B^{k+1}]^{\alpha_j}; A \rightsquigarrow B^k]$ *in case (A.2)), all firings of B during this block complete their execution before the first enabling point in the next block.*

This case is illustrated in Fig. 35. Each block is of the form $[A \rightsquigarrow B^2]^{\alpha_j}; A \rightsquigarrow B$, where the maximum value of α_j is 2. Therefore, five firings of B have to run in parallel with three firings of A. So, we must have $5t_B \leq 3t_A$. The computed sequence (δ_j) in case (II) is similar to that of case (I) but with small increments.

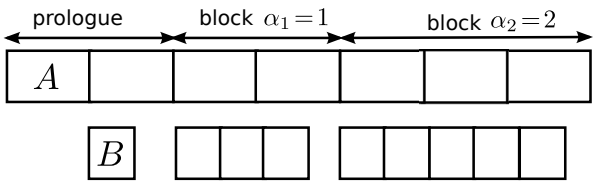


Figure 35: An ASAP execution of $A \xrightarrow{8\ \ 5} B$ with $t_A = 13$ and $t_B = 7$.

Patterns (A.1) and (B.1) are not possible in case (II).

Case (II), pattern (A.2):

Since $kt_B + \frac{r}{q}t_B \leq t_A < (k+1)t_B$ (*i.e.,* case (I) is not satisfied), we can put $t_A = kt_B + r'$. If $k+1$ firings of B are enabled, then there execution will be larger than t_A by $e = (k+1)t_B - t_A = t_B - r'$. According to the enabling pattern, a block is of the form $[A \rightsquigarrow B^{k+1}]^{\alpha_j}; A \rightsquigarrow B^k$. Hence, the firings of B brim over the block by $\alpha_j e + kt_B - t_A = \alpha_j(t_B - r') - r'$. Thus, to satisfy case (II), we must have $\forall j.\ \alpha_j(t_B - r') - r' \leq 0$. But, the maximum value of α_j is $\left\lceil \frac{r}{q-r} \right\rceil$. Therefore,

$$t_A \geq kt_B + \frac{\left\lceil \frac{r}{q-r} \right\rceil}{\left\lceil \frac{r}{q-r} \right\rceil + 1} t_B$$

Similarly to case (I.A.1), we compute sequence (δ_i) (see Fig. 36). Compared to case (I.A.2), the sequence is incremented by a $(+r)$. Hence, $\theta_{A,B} = \max_i \delta_i = 2p + q - \gcd(p,q) + r$.

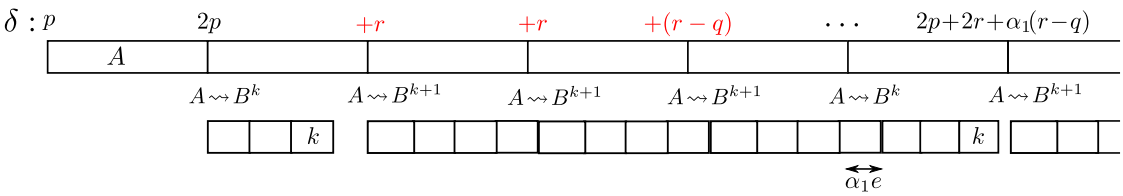


Figure 36: Case (II) pattern (A.2).

Case (II), pattern (A.3):

According to the enabling pattern, one block is of the form $A \rightsquigarrow B^{k+1}; [A \rightsquigarrow B^k]^{\beta_j}$. Hence, firings of B will brim over the block by $e + (kt_B - t_A)\beta_j$. Thus, to satisfy case (II), we must have $\forall j.\ e + (kt_B - t_A)\beta_j \leq 0$. But, the minimum value of β_j is $\left\lfloor \frac{q-r}{r} \right\rfloor$. Therefore,

$$t_A \geq kt_B + \frac{1}{\left\lfloor \frac{q}{r} \right\rfloor} t_B$$

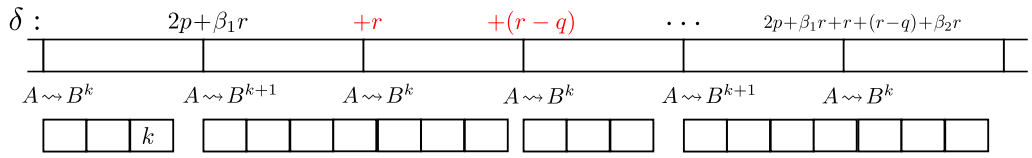


Figure 37: Case (II) pattern (A.3).

The sequence (δ_i) is illustrated in Fig. 37. Compared to case I.(A.3), it is incremented by $(+Xr)$ where X is the smallest integer such that $e+(kt_B-t_A)X \leq 0$ (*i.e.,* the overflow is caught up). So, $X = \left\lceil \frac{t_B}{t_A - kt_B} \right\rceil - 1$. Therefore, $\theta_{A,B} = 2p + q - \gcd(p,q) + Xr$.

Case (II), pattern (B.2):

Since $kt_A < t_B \leq kt_A + \frac{r}{p}t_A$ (*i.e.,* case (I) is not satisfied), we can put $t_B = kt_A + r'$. So, the execution time of one firing of B is larger than the execution time of k firings of A by $e = t_B - kt_A = r'$. According to the enabling pattern, a block is of the form $[A^k \rightsquigarrow B]^{\gamma_j}; A^{k+1} \rightsquigarrow B$. Hence, the firings of B brim over the block by $\gamma_j r' + t_B - (k+1)t_A$. Thus, to satisfy case (II), we must have $\forall j.\ \gamma_j r' + r' - t_A \leq 0$. But, the maximum value of γ_j is $\lceil \frac{p-r}{r} \rceil$. Therefore,

$$t_B \leq kt_A + \frac{1}{\lceil \frac{p}{r} \rceil} t_A$$

Similarly to case I.(B.2), we compute sequence (δ_i) (see Fig. 38). So, $\theta_{A,B} = 2(k+1)p + \max_j \left(j(p-r) - r \sum_{i=1}^{j} \gamma_i \right) = 2(k+1)p + r \max_j \left(\frac{jp}{r} - \lfloor \frac{jp}{r} \rfloor \right) = 2(k+1)p + (r - \gcd(p,q)) = 2p + 2q - r - \gcd(p,q)$.

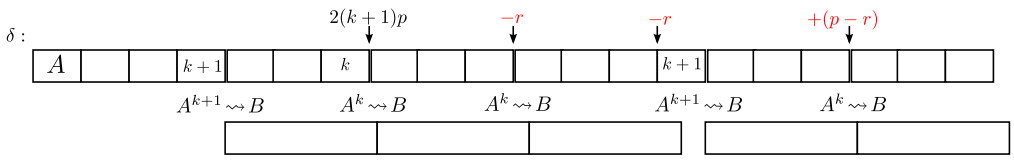


Figure 38: Case (II) pattern (B.2).

Case (II), pattern (B.3):

According to the enabling pattern, one block is of the form $A^k \rightsquigarrow B; [A^{k+1} \rightsquigarrow B]^{\lambda_j}$. Hence, firings of B will brim over the block by $t_B + \lambda_j t_B - kt_A - \lambda_j(k+1)t_A = r' + \lambda_j(r' - t_A)$. Thus, to satisfy case (II), we must have $\forall j.\ r' + \lambda_j(r' - t_A) \leq 0$. But, the minimum value of λ_j is $\lfloor \frac{r}{p-r} \rfloor$. Therefore, we must have

$$t_B \leq kt_A + \frac{\lfloor \frac{r}{p-r} \rfloor}{\lfloor \frac{r}{p-r} \rfloor + 1} t_A$$

The sequence (δ_i) is illustrated in Fig. 39. Compared to case I.(B.3), the sequence is incremented by $(+X(p-r))$ where X is the largest integer such that $r' + X(r' - t_A) > 0$ (*i.e.,* the overflow is not caught up). Hence, $\theta_{A,B} = p + q - \gcd(p,q) + \left\lceil \frac{t_B}{t_A} \right\rceil p + X(p-r)$. But, $\left\lceil \frac{t_B}{t_A} \right\rceil = k+1$. Therefore, $\theta_{A,B} = 2p + 2q - r - \gcd(p,q) + X(p-r)$. We have that $X = \left\lceil \frac{r'}{t_A - r'} \right\rceil - 1$.

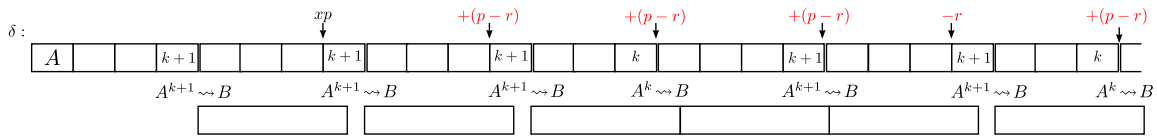


Figure 39: Case (II) pattern (B.3).

Case (III): *Otherwise.*

This is the most complicated case to solve since the sequence (x_j), which denotes the number of firings of B that have finished by the start of the jth firing of A, does not follow the enabling patterns. Our solution is based on the following observations. We define a catch-up sequence as a sequence of consecutive firings of B (*i.e.,* without gaps) that may spread over several blocks. Fig. 40 illustrates a catch-up sequence over two blocks.

The key observation is the following. For the firings of A inside a catch-up sequence, the number of firings of B that finish before firings of A actually follows the enabling pattern of graph $A \xrightarrow{t_A \quad t_B} B$, *i.e.,* as if *time* was produced and consumed instead of tokens. Furthermore, the maximum of sequence (δ_j) occurs inside the maximal (in terms of blocks) catch-up sequence. For instance, n in Eq. (37) represents the length of the maximal catch-up sequence.

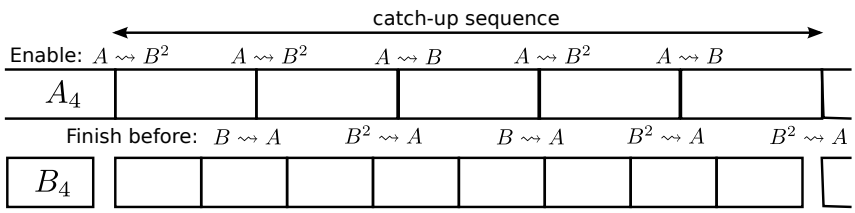


Figure 40: An ASAP execution of $A \xrightarrow{8 \quad 5} B$ with $t_A = 23$ and $t_B = 14$.

The obtained formulas are the following (patterns A.1 and A.2 are not possible):

Case III.

Case III.1. A.2

$$\theta_{A,B} = 2p + q + r - \gcd(p,q) + \max_{j=1}^{n-1}(jr \bmod (q-r)) \tag{37}$$

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 $r' = t_A - kt_B$.

Case III.2. A.3

$$\theta_{A,B} = 2p + 2q - \gcd(p,q) - \min_{j=1}^{n-1}(jq \bmod r) \tag{38}$$

where n is the smallest positive integer such that $\left\lceil \frac{nt_B}{r'} \right\rceil \leq \left\lfloor \frac{nq}{r} \right\rfloor$ and $r' = t_A - kt_B$.

Case III.3. B.2

$$\theta_{A,B} = 2p + 2q - r - \gcd(p,q) + \max_{j=1}^{n-1}(jp \bmod r) \tag{39}$$

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

Case III.4. B.3

$$\theta_{A,B} = 2p + 2q - \gcd(p,q) - \min_{j=1}^{n-1}(jr \bmod (p-r)) \tag{40}$$

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

Case (III), pattern (A.2):

Sequence (α_j) takes two values[4]; $\forall j.\ \alpha_j \in \left\{ \left\lfloor \frac{r}{q-r} \right\rfloor, \left\lceil \frac{r}{q-r} \right\rceil \right\}$. Once $\alpha_j = \left\lceil \frac{r}{q-r} \right\rceil$, an overflow will propagate from this block to the subsequent blocks. Thus, all firings of B till the catch-up point are consecutive. We call this sequence a catch-up sequence. The value of δ_i for the firing of A at the catch-up point will be equal to that computed in case II.(A.2). See Fig. 41.

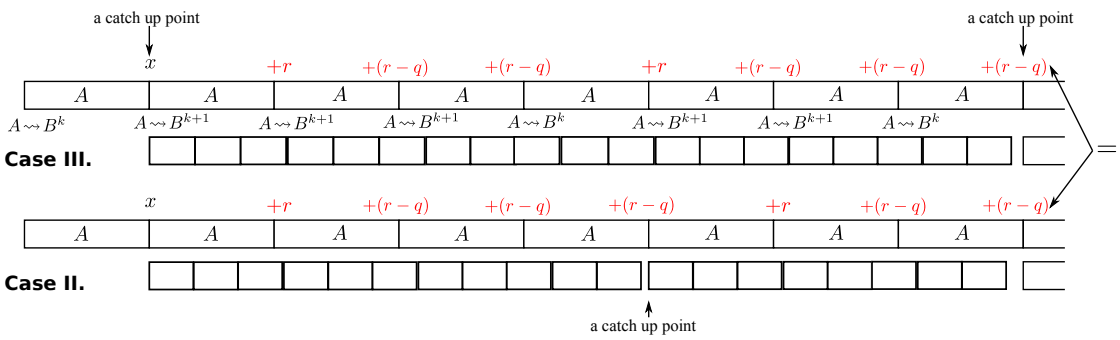


Figure 41: Case (III) pattern (A.2).

To know how many firings of B in the catch-up sequence finish before the start of any firing of A, we may look at the enabling pattern of the graph $A \xrightarrow{t_A \quad t_B} B$. Since $t_A \geq t_B$ and $t_B \leq 2r'$ (because A imposes a higher load than B), this enabling pattern is $\left[A \rightsquigarrow B^k; \left[A \rightsquigarrow B^{k+1}\right]^{\alpha'_j}\right]$ with $\alpha'_j = \left\lfloor \frac{jr'}{t_B - r'} \right\rfloor - \left\lfloor \frac{(j-1)r'}{t_B - r'} \right\rfloor$.

If a catch-up sequence starts at the jth block, then it will end at some block $(j+n-1)$ if and only if $\sum_{i=1}^{n} \alpha'_i \geq \sum_{i=j}^{j+n-1} \alpha_i$; *i.e.*, n is the smallest integer that satisfies

$$\lfloor nx_2 \rfloor \geq \lfloor (j-1)x_1 + nx_1 \rfloor - \lfloor (j-1)x_1 \rfloor \tag{41}$$

where $x_1 = \frac{r}{q-r}$ and $x_2 = \frac{r'}{t_B - r'}$. One important catch-up sequence is the longest one (called the maximal catch-up sequence). If we put $(j-1)r = a(q-r) + b$, then the previous equation can be rewritten as $\lfloor nx_2 \rfloor \geq \left\lfloor nx_1 + \frac{b}{q-r} \right\rfloor$. This implies that n is maximal when b is maximal; *i.e.*, $b = q - r - \gcd(q,r)$. However, this does not imply that the $(j-1)^{th}$ block such that $(j-1)r \bmod (q-r) = q - r - \gcd(p,q)$ is actually a catch-up point. To prove that we need to

[4]Note that if (α_j) ((β_j), (γ_j) or (λ_j)) takes only one value, then case (III) is impossible.

show that $\nexists m \leq (j-1).\ \sum_{i=1}^{m} \alpha'_i < \sum_{i=j-m}^{j-1} \alpha_i$; *i.e.*, $\lfloor mx_2 \rfloor < \lfloor (j-1)x_1 \rfloor - \lfloor (j-1)x_1 - mx_1 \rfloor$; which can be rewritten as $\lfloor mx_2 \rfloor < -\left\lfloor \frac{b}{q-r} - mx_1 \right\rfloor = \lfloor mx_1 \rfloor$. Inequality $\lfloor mx_2 \rfloor < \lfloor mx_1 \rfloor$ has no solution because $x_2 \geq x_1$ (since A imposes a higher load than B).

This means that among all catch-up points, the maximum value of (δ_i) occurs at the catch-up point before the maximal catch-up sequence; and it is equal to $2p + q - \gcd(p,q)$ (see Fig. 36). Furthermore, since any catch-up sequence is a prefix of the maximal catch-up sequence, the worst-case occurs inside this maximal sequence. Therefore, $\theta_{A,B} = 2p + q - \gcd(p,q) + (r + \max_{j=1}^{n-1}(rj + (r-q)\sum_{i=1}^{j} \alpha'_i)$ where n is the length of the maximal catch-up sequence. So, $\theta_{A,B} = 2p + q + r - \gcd(p,q) + \max_{j=1}^{n-1}(rj + (r-q)\lfloor jx_2 \rfloor)$.

But, according to Eq. (41), the maximal n is the smallest integer such that $\lfloor nx_2 \rfloor \geq \lceil nx_1 \rceil$ (recall that $(j-1)r \bmod (q-r) = q - r - \gcd(p,q)$). Hence, $\forall j < n.\ \lfloor jx_2 \rfloor < \lceil jx_1 \rceil$. But, we know that $\forall j.\ jx_2 \geq jx_1$ (because $x_2 \geq x_1$). Thus, $\forall j < n.\ \lfloor jx_1 \rfloor \leq \lfloor jx_2 \rfloor < \lceil jx_1 \rceil$; thus $\lfloor jx_2 \rfloor = \lfloor jx_1 \rfloor$. Therefore,

$$\theta_{A,B} = 2p + q + r - \gcd(p,q) + \max_{j=1}^{n-1}(jr \bmod (q-r)) \tag{42}$$

Case II.(A.2) is a special case of this one where all catch-up sequences are of length one. In the worst-case scenario, the catch-up sequence consists of an entire iteration. This occurs when $qt_A = pt_B$; thus $x_2 = x_1$. So, n is the smallest integer that satisfies $\lfloor nx_1 \rfloor \geq \lceil nx_1 \rceil$; thus $n = \frac{q-r}{\gcd(p,q)}$ (*i.e.*, an entire iteration). In this case, $\theta_{A,B} = 2p + q + r - \gcd(p,q) + \max_{j=1}^{\frac{q-r}{\gcd(p,q)}-1}(jr \bmod (q-r)) = 2(p + q - \gcd(p,q))$.

Case (III), pattern (A.3):

The proof follows the same scheme as that of case III.(A.2). Sequence (β_j) takes two value; $\forall j.\ \beta_j \in \left\{\left\lfloor \frac{q-r}{r} \right\rfloor, \left\lceil \frac{q-r}{r} \right\rceil\right\}$. Once $\beta_j = \left\lfloor \frac{q-r}{r} \right\rfloor$, an overflow will propagate from this block to the subsequent blocks.

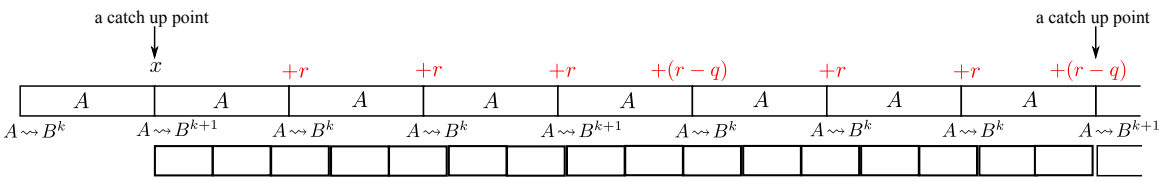


Figure 42: Case (III) pattern (A.3).

To know how many firings of B in the catch-up sequence finish before the start of any firing of A, we may look at the enabling pattern of the graph $A \xrightarrow{t_A \quad t_B} B$. Since $t_A \geq t_B$ and $t_B > 2r'$, this enabling pattern is $\left[\left[A \rightsquigarrow B^k\right]^{\beta'_j}; A \rightsquigarrow B^{k+1}\right]$ where $\beta'_j = \lceil jx_2 \rceil - \lceil (j-1)x_2 \rceil - 1$ with $x_2 = \frac{t_B}{r'}$. We also put $x_1 = \frac{q}{r}$.

If a catch-up sequence starts at the jth block, then it will end at some block $(j + n - 1)$ if

and only if $\sum_{i=1}^{n} \beta'_i \leq \sum_{i=j}^{j+n-1} \beta_i$; *i.e.*, n is the smallest integer that satisfies

$$\lceil nx_2 \rceil \leq \lceil (j-1)x_1 + nx_1 \rceil - \lceil (j-1)x_1 \rceil \tag{43}$$

If $(j-1)q = ar + b$, then the previous equation can be rewritten as $\lceil nx_2 \rceil + \lceil \frac{b}{r} \rceil \leq \lceil nx_1 + \frac{b}{r} \rceil$. This implies that n is maximal when b is minimal and not null; *i.e.*, $b = \gcd(p,q)$. However, this does not imply that the $(j-1)^{th}$ block such that $(j-1)r \bmod r = \gcd(p,q)$ is actually a catch-up point. To prove that we need to show that $\nexists m \leq (j-1).\ \sum_{i=1}^{m} \beta'_i > \sum_{i=j-m}^{j-1} \beta_i$; *i.e.*, $\lceil mx_2 \rceil > \lceil (j-1)x_1 \rceil - \lceil (j-1)x_1 - mx_1 \rceil$; which can be rewritten as $\lceil mx_2 \rceil > \left\lfloor mx_1 - \frac{\gcd(p,q)}{r} \right\rfloor + 1 = \lceil mx_1 \rceil$. But, this is impossible because $x_2 \leq x_1$.

If a catch-up point occurs at the jth block, then the value of δ_i is equal to $2p + \beta_1 r + \sum_{i=2}^{j} (\beta_1 r + (r-q)) = 2p + (q-r) + \sum_{i=1}^{j} (\beta_i r + (r-q)) = 2p + (q-r) + r(\lceil jx_1 \rceil - jx_1)$. Its maximum occurs when $jq \bmod r = \gcd(p,q)$; hence at the catch-up point before the maximal catch-up point; and it is equal to $2p + q - \gcd(p,q)$. Furthermore, since any catch-up sequence is a prefix of the maximal catch-up sequence, the worst-case occurs inside this maximal sequence. Therefore, $\theta_{A,B} = 2p + q - \gcd(p,q) + (\max_{j=1}^{n} \left(j(r-q) + r \sum_{i=1}^{j} \beta'_i) + (q-r) \right) = 2p + 2q - r - \gcd(p,q) + r \max_{j=1}^{n} (\lceil jx_2 \rceil - jx_1)$.

According to Eq. (43), the maximal n is the smallest integer such that $\lceil nx_2 \rceil \leq \lfloor nx_1 \rfloor$. Hence, $\forall j < n.\ \lceil jx_2 \rceil > \lfloor jx_1 \rfloor$. but, we know that $\forall j.\ jx_2 \leq jx_1$ (because $x_2 \leq x_1$). Therefore, $\forall j < n.\ \lceil jx_2 \rceil = \lceil jx_1 \rceil$. When $j = n$, we have that $\lceil jx_2 \rceil - jx_1 \leq 0$; hence it can be excluded from the equation. So,

$$\boxed{\theta_{A,B} = 2p + 2q - \gcd(p,q) - \min_{j=1}^{n-1} (jq \bmod r)} \tag{44}$$

In the worst-case scenario, the maximal catch-up sequence consists of an entire iteration. This occurs when $qt_A = pt_B$. Indeed, in this case, we have that $x_2 = x_1$. So, n is the smallest integer that satisfies $\lceil nx_1 \rceil \leq \lfloor nx_1 \rfloor$; thus $n = \frac{r}{\gcd(p,q)}$. Therefore, $\theta_{A,B} = 2p + 2q - \gcd(p,q) - \min_{j=1}^{\frac{r}{\gcd(p,q)}-1} (jq \bmod r) = 2(p + q - \gcd(p,q))$.

Case (III), pattern (B.2):

Sequence (γ_j) takes two values; $\forall j.\ \gamma_j \in \left\{ \left\lfloor \frac{p-r}{r} \right\rfloor, \left\lceil \frac{p-r}{r} \right\rceil \right\}$. Once $\gamma_j = \left\lceil \frac{p-r}{r} \right\rceil$, an overflow will propagate from this block to the subsequent ones. If a catch-up sequence starts at the jth block, then it will end at some block $(j+n-1)$ if and only if $\sum_{i=1}^{n} \gamma'_i \geq \sum_{i=j}^{j+n-1} \gamma_i$ with $\gamma'_j = \left\lfloor \frac{jt_A}{r'} \right\rfloor - \left\lfloor \frac{(j-1)t_A}{r'} \right\rfloor - 1$ where $r' = t_B - kt_A$. So, n is the smallest integer that satisfies

$$\lfloor nx_2 \rfloor \geq \lfloor (j-1)x_1 + nx_1 \rfloor - \lfloor (j-1)x_1 \rfloor \tag{45}$$

where $x_1 = \frac{p}{r}$ and $x_2 = \frac{t_A}{r'}$. If we put $(j-1)p = ar + b$, then the previous equation can be rewritten as $\lfloor nx_2 \rfloor \geq \left\lfloor nx_1 + \frac{b}{r} \right\rfloor$. This implies that n is maximal when b is maximal; *i.e.*, $b = r - \gcd(p,q)$.

We need to prove that the $(j-1)^{th}$ block such that $(j-1)p \bmod p = r - \gcd(p,q)$ is actually a catch-up point. To prove that, we need to show that $\nexists m \leq (j-1).\ \sum_{i=1}^{m} \gamma'_i < \sum_{i=j-m}^{j-1} \gamma_i$;

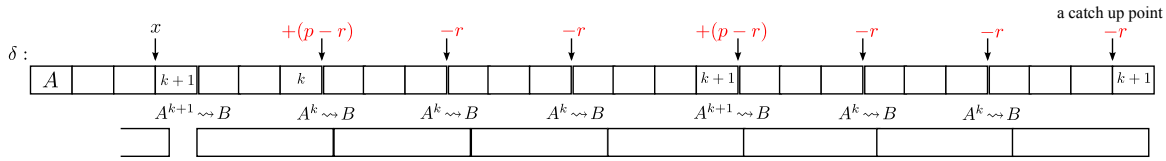


Figure 43: Case (III) pattern (B.2).

i.e., $\lfloor mx_2 \rfloor < \lceil mx_1 + \frac{\gcd(p,q)}{r} \rceil - 1 = \lfloor mx_1 \rfloor$. But, this is impossible because $x_2 \geq x_1$.

If a catch-up point occurs at the jth block, then the value of δ_i is equal to $2(k+1)p + \sum_{i=1}^{j}((p-r)-\gamma_i r)+(r-p) = p+2q-r+r(jx_1-\lfloor jx_1 \rfloor)$. Its maximum occurs when $jp \bmod r = r - \gcd(p,q)$ (*i.e.*, at the block before the maximal catch-up sequence) and it is equal to $p+2q-\gcd(p,q)$. Since any catch-up sequence is a prefix of the maximal catch-up sequence, we have that $\theta_{A,B} = p+2q-\gcd(p,q)+(p-r)+\max_{j=1}^{n-1}\left(j(p-r) - r\sum_{i=1}^{j}\gamma'_i\right) = 2p+2q-r-\gcd(p,q)+r\max_{j=1}^{n-1}(jx_1-\lfloor jx_2 \rfloor)$.

According to Eq. (45), the maximal n is the smallest integer such that $\lfloor nx_2 \rfloor \geq \lceil nx_1 \rceil$. Hence, $\forall j < n.\ \lfloor jx_2 \rfloor < \lceil jx_1 \rceil$. But, we know that $\forall j.\ jx_2 \geq jx_1$ (because $x_2 \geq x_1$). Thus, $\forall j < n.\ \lfloor jx_2 \rfloor < \lfloor jx_1 \rfloor$. Therefore,

$$\theta_{A,B} = 2p + 2q - r - \gcd(p,q) + \max_{j=1}^{n-1}(jp \bmod r) \tag{46}$$

In the worst-case scenario (*i.e.*, $x_2 = x_1$), n is the smallest integer such that $\lfloor nx_1 \rfloor \geq \lceil nx_1 \rceil$. Thus, $n = \frac{r}{\gcd(p,q)}$. Therefore, $\theta_{A,B} = 2p + 2q - r - \gcd(p,q) + \max_{j=1}^{\frac{r}{\gcd(p,q)}-1}(jp \bmod r) = 2(p+q-\gcd(p,q))$.

Case (III), pattern (B.3):

Sequence (λ_j) takes two values; $\forall j.\ \lambda_j \in \left\{\lfloor \frac{r}{p-r} \rfloor, \lceil \frac{r}{p-r} \rceil\right\}$. Once $\lambda_j = \left\lfloor \frac{r}{p-r} \right\rfloor$, an overflow will propagate from this block to the subsequent ones. If a catch-up sequence starts at the jth block, then it will end at some block $(j+n-1)$ if and only if $\sum_{i=1}^{n}\lambda'_i \leq \sum_{i=j}^{j+n-1}\lambda_i$ with $\lambda'_j = \left\lceil \frac{jr'}{t_A - r'} \right\rceil - \left\lceil \frac{(j-1)r'}{t_A - r'} \right\rceil$. So, n is the smallest integer that satisfies

$$\lceil nx_2 \rceil \leq \lceil (j-1)x_1 + nx_1 \rceil - \lceil (j-1)x_1 \rceil \tag{47}$$

where $x_1 = \frac{r}{p-r}$ and $x_2 = \frac{r'}{t_A - r'}$. If we put $(j-1)r = a(p-r)+b$, then the previous equation can be rewritten as $\lceil nx_2 \rceil + \left\lceil \frac{b}{p-r} \right\rceil \leq \left\lceil nx_1 + \frac{b}{p-r} \right\rceil$. This implies that n is maximal when b is minimal but not null; *i.e.*, $b = \gcd(p,r)$. However, this does not imply that the $(j-1)^{th}$ block such that $(j-1)r \bmod (p-r) = \gcd(p,q)$ is actually a catch-up point. To prove that we need to show that $\nexists m \leq (j-1).\ \sum_{i=1}^{m}\lambda'_i > \sum_{i=j-m}^{j-1}\lambda_i$; *i.e.*, $\lceil mx_2 \rceil > 1 + \left\lfloor mx_1 - \frac{\gcd(p,q)}{p-r} \right\rfloor = \lceil mx_1 \rceil$. But, this is impossible because $x_2 \leq x_1$.

If a catch-up point occurs at the jth block, then the value of δ_i is equal to $2(k+1)p + (\lambda_1 - 2)(p-r) + \sum_{i=2}^{j}(\lambda_i(p-t)-r) = 2q + r + (p-r)(\lceil jx_1 \rceil - jx_1)$. Its maximum occurs when

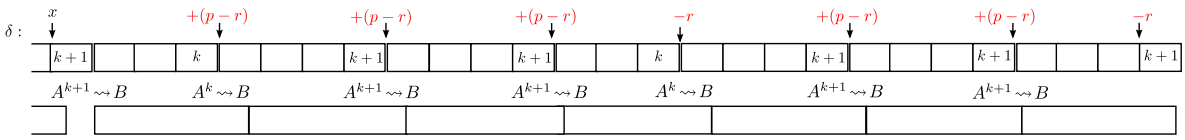


Figure 44: Case (III) pattern (B.3).

$jr \bmod (p-r) = \gcd(p,q)$ (*i.e.,* just before the maximal catch-up sequence) and it is equal to $2q+p-\gcd(p,q)$. Since any catch-up sequence is a prefix of the maximal catch-up sequence, we have that $\theta_{A,B} = 2q+p-\gcd(p,q) + (r + \max_{j=1}^{n}\left(\sum_{i=1}^{j}(\lambda'_i(p-r)-r)\right)) = 2q+p+r-\gcd(p,q) + (p-r)\max_{j=1}^{n}(\lceil jx_2\rceil - jx_1)$.

According to Eq. (46), n is the smallest integer such that $\lceil nx_2\rceil \leq \lfloor nx_1\rfloor$. Hence, $\forall j < n.\ \lceil jx_2\rceil > \lfloor jx_1\rfloor$. But, we know that $\forall j.\ jx_2 \leq jx_1$ (because $x_2 \leq x_1$). Thus, $\forall j < n.\ \lceil jx_1\rceil = \lceil jx_1\rceil$. When $j = n$, we have that $\lceil jx_2\rceil - jx_1 \leq 0$; hence it can be excluded from the equation. So,

$$\theta_{A,B} = 2p + 2q - \gcd(p,q) - \min_{j=1}^{n-1}(jr \bmod (p-r)) \tag{48}$$

□

Proof. (**Property 4.3**) The minimal buffer size of G^{-1} for maximal throughput is $\theta_{B,A}$. That is, the graph $B \xrightarrow{q\ \ p} A$ with a backward edge $A \xrightarrow{p\ \ q} B$ with $\theta_{B,A}$ initial tokens, achieves the maximal throughput. Prop. 3.1 ensures that the dual graph (and therefore G) have the same throughput. This throughput is also maximal since if there exists another number of tokens allowing a better throughput for G, by duality it would also represent a better throughput for G^{-1}: a contradiction. Similarly, this number of tokens is minimal since if there exists $\theta_{A,B} < \theta_{B,A}$ allowing the same maximal throughput, by duality $\theta_{A,B}$ would also achieve a maximal throughput for G^{-1} which contradicts the minimality of $\theta_{B,A}$. Therefore, the minimal buffer size for G needed to achieve maximal throughput is equal to $\theta_{B,A}$. □

Proof. (**Property 4.4**)

The initial buffer state of the graph is $s_0 = \begin{pmatrix} d \\ \theta'_{A,B} \end{pmatrix}$. Firstly, we prove that there is a **live** schedule (which consists of x firings of A and y firings of B) that transforms the initial state s_0 to state $s_1 = \begin{pmatrix} d \bmod \gcd(p,q) \\ \theta'_{A,B} + d - d \bmod \gcd(p,q) \end{pmatrix}$.

Though equation $d + xp - yq = d \bmod \gcd(p,q)$ is a feasible equation [Using Bézout identity], this does not imply the existence of a live schedule. The schedule is constructed as follows. Initially, there are $\sigma_0 = d$ tokens in the channel $A \to B$. After firing B as much as possible, the number of remaining tokens is $\sigma_1 = d \bmod q$. Since the graph is live, A can fire at least once. Then, actor B is fired as much as possible; and the number of remaining tokens is $\sigma_2 =$

$(\sigma_1 + p) \bmod q$. The process can be repeated infinitely; and the sequence of number of remaining tokens is $\sigma_n = (\sigma_{n-1} + p) \bmod q$. Since $(a \bmod b + c) \bmod b = (a + c) \bmod b$, we can put $\sigma_n = (d + (n-1)p) \bmod q$. The question now is whether there exists n such that $\sigma_n = d \bmod \gcd(p, q)$. The answer is yes [using BÃ©zout identity]. So, state s_1 is reachable from state s_0. Therefore, if the ASAP execution does not achieve the maximum throughput starting from s_1, then it cannot do that starting from s_0.

Secondly, for a graph with $d \bmod \gcd(p, q)$ on the forward edge, $\theta_{A,B}$ tokens on the backward edge are still needed to achieve the maximum throughput. This follows immediately from property B.1. The normalized channel will contain zero token since $d \bmod \gcd(p, q) < \gcd(p, q)$.

Hence, we must have $\theta'_{A,B} + d - d \bmod \gcd(p, q) \geq \theta_{A,B}$. □

Property B.1 (Canonical form)**.** Let $p' = \frac{p}{\gcd(p,q)}$ and $q' = \frac{q}{\gcd(p,q)}$. Replacing any channel $A \xrightarrow{p \quad q} B$ with d initial tokens in the graph by a channel $A \xrightarrow{p' \quad q'} B$ with $\left\lfloor \frac{d}{\gcd(p,q)} \right\rfloor$ initial tokens, does not alter the ASAP execution of the graph.

Proof. (**Property B.1**)

The case when $\gcd(p, q)$ divides d is trivial. The other case is less evident. Firstly, we know that the new rates need to have a ratio equal to $\frac{p'}{q'}$ (since i tends to infinity, any variation from this ratio will give eventually different results). So, we take the new rates as np' and nq' and we have to find the minimum values of $n, x \in \mathbb{N}$ such that

$$\forall i.\ \left\lceil \frac{iq - d}{p} \right\rceil = \left\lceil \frac{inq' - x}{np'} \right\rceil \tag{49}$$

Eq. 49 means that the data-dependencies created by both channels are the same. But, we have $\left\lceil \frac{inq'-x}{np'} \right\rceil = \left\lceil \frac{iq-d}{p} + \frac{nd-x\gcd(p,q)}{np} \right\rceil$. Let $r_i = (iq - d) \bmod p$. We have that $\forall i.\ r_i \neq 0$ since $\gcd(p, q)$ does not divide d. Thus, Eq. 49 can be rewritten as $\forall i.\ 1 = \left\lceil \frac{r_i}{p} + \frac{nd-x\gcd(p,q)}{np} \right\rceil$; and hence $\forall i.\ 0 < \frac{r_i}{p} + \frac{nd-x\gcd(p,q)}{np} \leq 1$. So, we have $\frac{\max_i r_i + d - p}{\gcd(p,q)} \leq \frac{x}{n} < \frac{\min_i r_i + d}{\gcd(p,q)}$.

But, $\forall i.$ $\gcd(p, q)$ divides $r_i + d$ [Using Bézout identity]. Furthermore, we have that $\max_i r_i - \min_i r_i = p - \gcd(p, q)$ [Using Bézout identity]. So, $\frac{d + \min_i r_i}{\gcd(p,q)} - 1 \leq \frac{x}{n} < \frac{d + \min_i r_i}{\gcd(p,q)}$. Therefore, $\frac{x}{n}$ lies between two successive integers. Hence, the minimum value of n is 1, while x takes the value $\frac{d + \min_i r_i}{\gcd(p,q)} - 1$.

Since $\min_i r_i$ is the smallest integer that makes $\gcd(p, q)$ divides $d + \min_i r_i$, we put $x = \left\lfloor \frac{d}{\gcd(p,q)} \right\rfloor$. □

B.3 Multi-iteration latency

Proof. (**Property 4.5**) The three cases (I), (II) and (III) are those described in the previous section (see the proof of Property 4.2).

• **Case (I)**: All the firings of B that have been enabled before the last enabling point in the iteration (*i.e.,* the end of the last firing of A) will finish by that point. According to the enabling patterns (Fig. 8), the last firing of A enables $\lceil p/q \rceil$ firings of B. Hence, $\Delta_{A,B} = \lceil p/q \rceil\, t_B$.

• **Case (II)**: All the firings of B that have been enabled during one block will finish by the beginning of the next block. Hence, we have to consider the last block in each enabling pattern. There are four cases (A.2), (A.3), (B.2), and (B.3).

In case (A.2), the last block is of the form $A \rightsquigarrow B^k; \left[A \rightsquigarrow B^{k+1}\right]^{\lceil \frac{r}{q-r} \rceil}$. Therefore, $\Delta_{A,B} = \left\lceil \frac{r}{q-r} \right\rceil (k+1)t_B - \left(\left\lceil \frac{r}{q-r} \right\rceil - 1\right) t_A$.

In case (A.3), the last block is of the form $\left[A \rightsquigarrow B^k\right]^{\lfloor \frac{q-r}{r} \rfloor}; A \rightsquigarrow B^{k+1}$. Thus, $\Delta_{A,B} = (k+1)t_B$. Both cases (A.2) and (A.3) can be unified in Eq. (17).

In case (B.2), the last block is of the form $A^{k+1} \rightsquigarrow B; \left[A^k \rightsquigarrow B\right]^{\lceil \frac{p-r}{r} \rceil}$. Hence, $\Delta_{A,B} = \left(\lceil \frac{p-r}{r} \rceil + 1\right) t_B - \lceil \frac{p-r}{r} \rceil k t_A$.

In case (B.3), the last block is of the form $\left[A^{k+1} \rightsquigarrow B\right]^{\lfloor \frac{r}{p-r} \rfloor}; A^k \rightsquigarrow B$. Therefore, $\Delta_{A,B} = 2t_B - kt_A$. Both cases (B.2) and (B.3) can be unified in Eq. (18).

• **Case (III)**: The obtained formulas are:

Case III.

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

$$\Delta_{A,B} = t_A + r' + \frac{t_B r - qr'}{\gcd(p,q)} + (t_B - r') \max_{j=0}^{n-1} \left(\frac{jr'}{t_B - r'} - \left\lfloor \frac{jr}{q-r} \right\rfloor \right) \tag{50}$$

Case III.2. Let $r' = t_A - kt_B$ and $n = \frac{r}{\gcd(p,q)}$

$$\Delta_{A,B} = (k+1)t_B + \frac{t_B r - qr'}{\gcd(p,q)} + r' \max_{j=0}^{n-1} \left(\left\lceil \frac{jq}{r} \right\rceil - \frac{jt_B}{r'} \right) \tag{51}$$

Case III.3. Let $r' = t_B - kt_A$ and $n = \frac{r}{\gcd(p,q)}$

$$\Delta_{A,B} = (k+1)t_A + \frac{pr' - t_A r}{\gcd(p,q)} + r' \max_{j=0}^{n-1} \left(\frac{jt_A}{r'} - \left\lfloor \frac{jp}{r} \right\rfloor \right) \tag{52}$$

Case III.4. Let $r' = t_B - kt_A$ and $n = \frac{p-r}{\gcd(p,q)}$

$$\Delta_{A,B} = t_B + r' + \frac{pr' - t_A r}{\gcd(p,q)} + (t_A - r') \max_{j=0}^{n-1} \left(\left\lceil \frac{jr}{p-r} \right\rceil - \frac{jr'}{t_A - r'} \right) \tag{53}$$

Case (III), pattern (A.2):

Suppose that the last catch-up sequence in the iteration starts at the jth block. Thus, all the remaining firings of B are consecutive. Let us put $N = \frac{q-r}{\gcd(p,q)}$. The number of remaining firings of A is equal to $N - j - 1 + \sum_{i=j}^{N} \alpha_i$, while the number of remaining firings of B is equal to $k(N-j) + (k+1) \sum_{i=j}^{N} \alpha_i$. Therefore, $\Delta_{A,B} = t_A - r'(N-j) + (t_B - r') \sum_{i=j}^{N} \alpha_i$. If $x_1 = \frac{r}{q-r}$ and $x_2 = \frac{r'}{t_B - r'}$, then $\Delta_{A,B} = t_A + r' + \frac{t_B r - qr'}{\gcd(p,q)} + (t_B - r')((j-1)x_2 - \lfloor (j-1)x_1 \rfloor)$.

We can rewrite $(t_B - r')((j-1)x_2 - \lfloor (j-1)x_1 \rfloor)$ as $\sum_{i=1}^{j-1} (t_A + \alpha_i t_A) - \sum_{i=1}^{j-1} (kt_B + \alpha_i (k+1) t_B)$;

i.e., as the difference between the sum of execution times of A over the first $(j-1)$ blocks minus the sum of execution times of B over the same blocks. The maximum of this difference occurs at the last catch-up point before the end of the iteration. Therefore, we can put

$$\Delta_{A,B} = t_A + r' + \frac{t_B r - q r'}{\gcd(p,q)} + (t_B - r') \max_{i=0}^{N-1}(i x_2 - \lfloor i x_1 \rfloor)$$

So, $\Delta_{A,B}$ can be upper bounded by $t_A + r' + \frac{t_B r - q r'}{\gcd(p,q)} + (t_B - r') \max_{i=0}^{N-1}(i x_2 - i x_1) + (t_B - r') \max_{i=0}^{N-1}(i x_1 - \lfloor i x_1 \rfloor) = t_A + r' + \frac{t_B r - q r'}{\gcd(p,q)} + (t_B - r')(N-1)(x_2 - x_1) + (t_B - r')\frac{q - r - \gcd(p,q)}{q-r} = t_A + \frac{(t_B - r')(q - \gcd(p,q))}{q-r}$. Therefore,

$$\Delta_{A,B} \leq t_A + \frac{(t_B - r')(q - \gcd(p,q))}{q-r}$$

In the worst-case scenario (*i.e.,* when $q t_A = p t_B$), we have that $\frac{t_B - r'}{q-r} = \frac{t_B}{q}$ and hence $\Delta_{A,B} = \frac{t_B}{q}(p + q - \gcd(p,q))$.

Case (III), pattern (A.3):

Similarly to case (A.2), we suppose that the last catch-up sequence in the iteration starts at the jth block and we put $N = \frac{r}{\gcd(p,q)}$. The number of remaining firings of A is equal to $(N-j+1) + \sum_{i=j}^{N} \beta_i$, while the number of remaining firings of B is equal to $(k+1) + \sum_{i=j}^{N}(k\beta_i + (k+1))$. Therefore, $\Delta_{A,B} = (k+1)t_B + (N-j+1)(t_B - r') - r' \sum_{i=j}^{N} \beta_i$.

If $x_1 = \frac{q}{r}$ and $x_2 = \frac{t_B}{r'}$, then $\Delta_{A,B} = (k+1)t_B + \frac{t_B r - q r'}{\gcd(p,q)} + r'(\lceil (j-1)x_1 \rceil - (j-1)x_2)$. Hence,

$$\Delta_{A,B} = (k+1)t_B + \frac{t_B r - q r'}{\gcd(p,q)} + r' \max_{i=0}^{N-1}(\lceil i x_1 \rceil - i x_2)$$

Hence, $\Delta_{A,B} \leq (k+1)t_B + \frac{t_B r - q r'}{\gcd(p,q)} + r' \max_{i=0}^{N-1}(\lceil i x_1 \rceil - i x_1) + r' \max_{i=0}^{N-1}(i x_1 - i x_2)$. So,

$$\Delta_{A,B} \leq t_A + 2t_B - \frac{r'}{r}(q + \gcd(p,q))$$

Case (III), pattern (B.2):

Suppose that the last catch-up sequence in the iteration starts at the jth block. Let us put $N = \frac{r}{\gcd(p,q)}$. The number of remaining firings of A is equal to $(N-j)(k+1) + \sum_{i=j}^{N} k\gamma_j$, while the number of remaining firings of B is equal to $(N-j+1) + \sum_{i=j}^{N} \gamma_j$. Therefore, $\Delta_{A,B} = t_B - (N-j)(t_A - r') + r' \sum_{i=j}^{N} \gamma_j$. If $x_2 = \frac{t_A}{r'}$ and $x_2 = \frac{p}{r}$, then

$$\Delta_{A,B} = (k+1)t_A + \frac{p r' - t_A r}{\gcd(p,q)} + r' \max_{i=0}^{N-1}(i x_2 - \lfloor i x_1 \rfloor)$$

So, $\Delta_{A,B}$ can be upper bounded by $(k+1)t_A + \frac{p r' - t_A r}{\gcd(p,q)} + r' \max_{i=0}^{N-1}(i x_1 - \lfloor i x_1 \rfloor) + r' \max_{i=0}^{N-1}(i x_2 - i x_1)$. Hence,

$$\Delta_{A,B} \leq t_B + \frac{r'}{r}(p - \gcd(p,q))$$

Case (III), pattern (B.3):

Again, we suppose that the last catch-up sequence in the iteration starts at the jth block and we put $N = \frac{p-r}{\gcd(p,q)}$. The number of remaining firings of A is equal to $(N-j+2)k + \sum_{i=j}^{N} \lambda_i(k+1)$, while the number of remaining firings of B is equal to $(N-j+3) + \sum_{i=j}^{N} \lambda_i$. Therefore, $\Delta_{A,B} = t_B + (N-j+2)r' - (t_A - r') \sum_{i=j}^{N} \lambda_i$. If $x_2 = \frac{r'}{t_A - r'}$ and $x_2 = \frac{r}{p-r}$, then

$$\Delta_{A,B} = t_B + r' + \frac{pr' - t_A r}{\gcd(p,q)} + (t_A - r') \max_{i=0}^{N-1}(\lceil ix_1 \rceil - ix_2)$$

Hence,

$$\Delta_{A,B} \leq t_A + t_B + r' - \frac{t_A - r'}{p-r}(r + \gcd(p,q))$$

□

C Linearization of $A \xrightarrow{p \quad q} B$

C.1 Forward linearization

Proof. (Forward lower and upper bound linearizations) We will show only one case, namely case (I) with pattern (A.2). In this case, since $z_A t_A \geq z_B t_B$, we have $t_{B^s} = t_{B^\ell} = \frac{q t_A}{p}$, $t^0_{B^s} = \max_i(f_B(i) - i t_{B^s})$, and $t^0_{B^\ell} = \min_i(f_B(i) - i t_{B^\ell})$. So, to compute the upper bound and the lower bound linearizations, we need to compute the maximum and the minimum of sequence $(f_B(i) - i\frac{q t_A}{p})$ respectively.

Case (I):

Case (I), pattern (A.1):

We have that $f_B(ik+j) = (i+1)t_A + j t_B$ where $1 \leq j \leq k$. Hence, $t^0_{B^s} = \max_{1 \leq j \leq k}((i+1)t_A + j t_B - (ik+j)t_{B^s}) = \max_{1 \leq j \leq k}\left(t_A + j(t_B - \frac{t_A}{k})\right)$. Since $t_B - \frac{t_A}{k} \leq 0$, we have $t^0_{B^s} = t_A + t_B - \frac{1}{k}t_A$.

We also have $t^0_{B^\ell} = \min_{1 \leq j \leq k}\left(t_A + j(t_B - \frac{t_A}{k})\right) = k t_B$.

Case (I), pattern (A.2):

For a batch of consecutive firings of B, the maximum of function $f_B(i) - i t_{B^s}$ occurs at the first firing. The sequence of values that takes this function (for the first firing of each batch) is illustrated in Fig. 45 where $x = t_A + t_B - t_{B^s}$, $\sigma_1 = t_A - k t_{B^s}$, and $\sigma_2 = t_A - (k+1)t_{B^s}$. We have that $\sigma_1 = \frac{r t_A}{p} > 0$ and $\sigma_2 = \frac{(r-q)t_A}{p} < 0$.

So, $t^0_{B^s} = x + \sigma_1 + \max_j\left(j\sigma_1 + \sigma_2 \sum_{i=1}^{j} \alpha_i\right) = x + \sigma_1 + \frac{(q-r)t_A}{p} \max_j\left(\frac{jr}{q-r} - \left\lfloor \frac{jr}{q-r} \right\rfloor\right) = x + \sigma_1 + \frac{t_A}{p}(q - r - \gcd(p,q))$. Therefore, $t^0_{B^s} = t_A + t_B - \frac{t_A}{p}\gcd(p,q)$.

Similarly, to compute $t^0_{B^\ell}$ we need to consider the last firing in each batch of consecutive firings of B. The minimum of the constructed sequence is $t_B + \min\{k t_B + \sigma_1 + \min_j\left(j\sigma_1 + \sigma_2 \sum_{i=1}^{j} \alpha_i\right), (k+1)t_B + \min_j\left(j\sigma_1 + \sigma_2 \sum_{i=1}^{j} \alpha_i\right)\} = t_B + \min\{k t_B + \sigma_1, (k+1)t_B\}$. Hence, $t^0_{B^\ell} = k t_B + \min\left\{\frac{r t_A}{p}, t_B\right\}$.

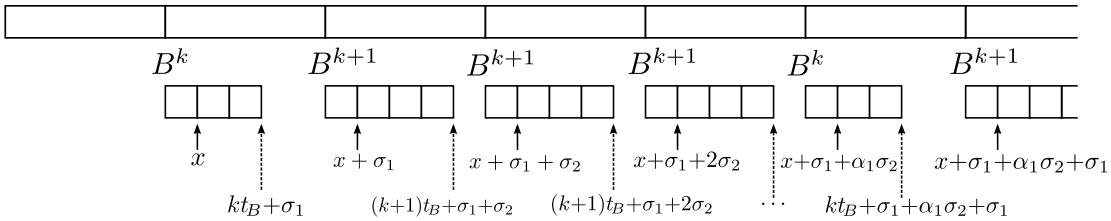


Figure 45: Case (I) pattern (A.2): values of $f_B(i) - it_{B^s}$.

Case (I), pattern (A.3):

Similarly to case (I) pattern (A.2), we construct the sequence $f_B(i) - it_{B^s}$. See Fig. 46 where $x = t_A + t_B - t_{B^s}$, $\sigma_1 = t_A - kt_{B^s} = \frac{rt_A}{p} > 0$, and $\sigma_2 = t_A - (k+1)t_{B^s} = \frac{(r-q)t_A}{p} < 0$.

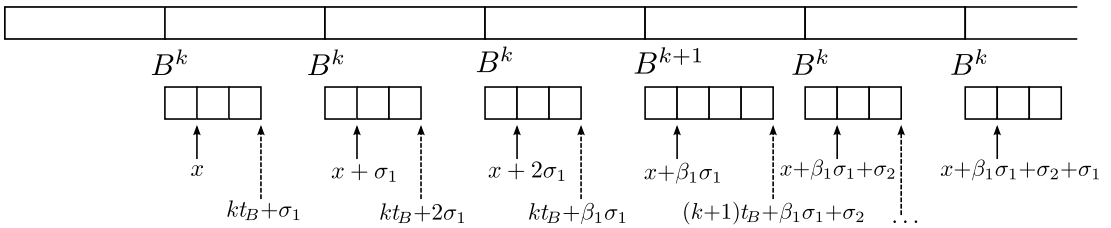


Figure 46: Case (I) pattern (A.3): values of $f_B(i) - it_{B^s}$.

Thus, $t^0_{B^s} = x - \sigma_2 + \max_j \left(j\sigma_2 + \sigma_1 \sum_{i=1}^{j} \beta_i \right) = x - \sigma_2 + \frac{rt_A}{p} \max_j \left(\lceil \frac{jq}{r} \rceil - \frac{jq}{r} \right) = x - \sigma_2 + \frac{t_A}{p}(r - \gcd(p,q))$. Thus, $t^0_{B^s} = t_A + t_B - \frac{t_A}{p}\gcd(p,q)$.

To compute $t^0_{B^\ell}$ we need to consider the last firing in each batch of consecutive firings of B. So, $t^0_{B^\ell} = \min \left\{ kt_B + \sigma_1 + \min_j \left(j\sigma_2 + \sigma_1 \sum_{i=1}^{j} \beta_i \right), (k+1)t_B + \min_j \left(j\sigma_2 + \sigma_1 \sum_{i=1}^{j} \beta_i \right) \right\} = \min\{kt_B + \sigma_1, (k+1)t_B\}$. Hence, $t^0_{B^\ell} = kt_B + \min\left\{ \frac{rt_A}{p}, t_B \right\}$.

Case (I), pattern (B.1):

We have that $f_B(i) = ikt_A + t_B$. Therefore, $t^0_{B^s} = \max_i(ikt_A + t_B - it_{B^s}) = t_B$; while $t^0_{B^\ell} = t_B + \min_i(ikt_A - it_{B^s}) = t_B$.

Case (I), pattern (B.2):

The sequence of values of $f_B(i) - it_{B^s}$ is illustrated in Fig. 47 where $\sigma_1 = kt_A - t_{B^s}$ and $\sigma_2 = (k+1)t_A - t_{B^s}$.

We have that $\sigma_1 = \frac{-rt_A}{p} < 0$ and $\sigma_2 = \frac{(p-r)}{p}t_A > 0$. Thus, $t^0_{B^s} = t_B + \sigma_2 + \max_j \left(j\sigma_2 + \sigma_1 \sum_{i=1}^{j} \gamma_i \right) = t_B + \sigma_2 + \frac{rt_A}{p} \max_j \left(\frac{jp}{r} - \lfloor \frac{jp}{r} \rfloor \right) = t_B + \sigma_2 + \frac{t_A}{p}(r - \gcd(p,q))$. Thus, $t^0_{B^s} = t_A + t_B - \frac{t_A}{p}\gcd(p,q)$.

We also have $t^0_{B^\ell} = t_B + \min_j \left(j\sigma_2 + \sigma_1 \sum_{i=1}^{j} \gamma_i \right) = t_B$.

Case (I), pattern (B.3):

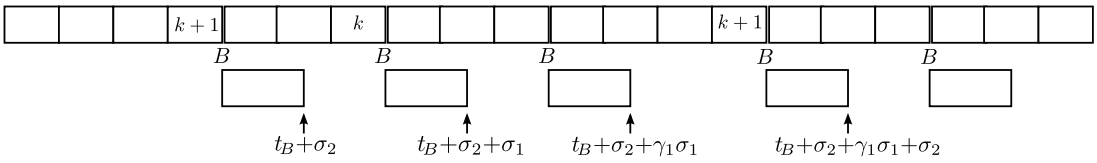


Figure 47: Case (I) pattern (B.2): values of $f_B(i) - it_{B^s}$.

Similarly to case (B.2), the constructed sequence is illustrated in Fig. 48 where $\sigma_1 = kt_A - t_{B^s}$ and $\sigma_2 = (k+1)t_A - t_{B^s}$.

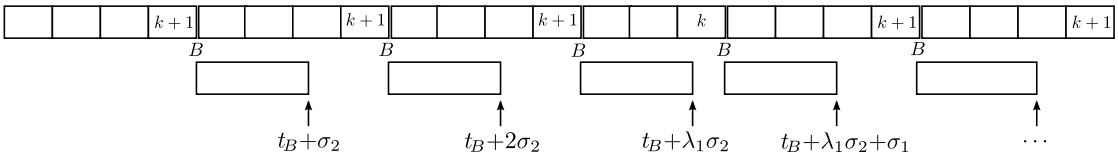


Figure 48: Case (I) pattern (B.3): values of $f_B(i) - it_{B^s}$.

Therefore, $t^0_{B^s} = t_B - \sigma_1 + \max\limits_j \left(j\sigma_1 + \sigma_2 \sum\limits_{i=1}^{j} \lambda_i \right) = t_B - \sigma_1 + \frac{(p-r)t_A}{p} \max\limits_j \left(\left\lceil \frac{jr}{p-r} \right\rceil - \frac{jr}{p-r} \right) = t_B - \sigma_1 + \frac{t_A}{p}(p - r - \gcd(p,q))$. Hence, $t^0_{B^s} = t_A + t_B - \frac{t_A}{p}\gcd(p,q)$.

We also have $t^0_{B^\ell} = t_B + \min\limits_j \left(j\sigma_1 + \sigma_2 \sum\limits_{i=1}^{j} \lambda_i \right) = t_B$.

Case (II)+(III):

pattern (A.2):

As illustrated in Fig. 49, firings of B in a catch-up sequence are consecutive and the maximum of $f_B(i) - it_{B^s}$ occurs at the first firing (since both $\sigma'_1 = kt_B - kt_{B^s}$ and $\sigma'_2 = (k+1)t_B - (k+1)t_{B^s}$ are non-positive). Thus, it is sufficient to look only at the first firing after each catch-up point. At these points, the value of $f_B(i) - it_{B^s}$ is equal to that of case (I) pattern (A.2) in which the maximum occurs when $jr \mod (q-r) = q - r - gcd(p,q)$; hence at the beginning of the maximal catch-up sequence. Hence, Eq. (21) is also valid in this case.

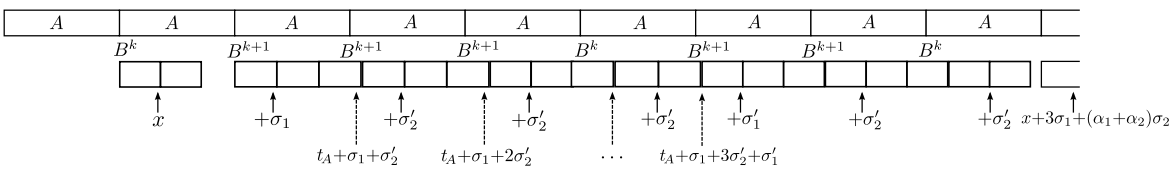


Figure 49: Case (III) pattern (A.2): values of $f_B(i) - it_{B^s}$.

Since both σ'_1 and σ'_2 are non-positive, the minimum of $f_B(i) - it_{B^s}$ occurs at the last firing in a batch of consecutive firings. Therefore, it is sufficient to look only at the last firing of each catch-up sequence. But, as the start of the last B^k of the catch-up sequence is not synchronous with the start of a firing of A (unlike in case (I) pattern (A.2)), $f_B(i) - it_{B^s}$ computed in this

case is larger than or equal to what is computed by the formula of case (I) pattern (A.2); *i.e.,* $kt_B + \sigma_1 + \left(j\sigma_1 + \left\lfloor \frac{jr}{q-r} \right\rfloor \sigma_2\right)$. So, this equation can be taken as a lower bound. However, since its minimum ($kt_B + \sigma_1$) occurs also in case III.A.2 (see the first B^k in Fig. 49), the solution of case (I) pattern (A.2) is also exact in this case.

pattern (A.3):

In case (I) pattern (A.3), the maximum occurs when $jq \mod r = \gcd(p, q)$; *i.e.,* at the beginning of the maximal catch-up sequence. Therefore, Eq. (21) is also valid in this case. Furthermore, the minimum of $kt_B + \sigma_1 + \min_j \left(j\sigma_2 + \sigma_1 \sum_{i=1}^{j} \beta_i\right)$ (Case (I) pattern (A.3)) also occurs in this case (for the first B^k). Hence, the solution of case (I) (A.3) is also exact in this case.

pattern (B.2):

For a batch of consecutive firings of B, the maximum of $f_B(i) - it_{B^s}$ occurs at the first firing while its minimum occurs at the last firing. This is illustrated in Fig. 50 where $\sigma'_1 = t_B - t_{B^s} = \frac{pt_B - qt_A}{p} \leq 0$.

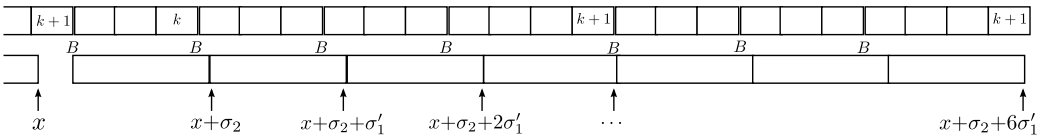


Figure 50: Case (III) pattern (B.2): values of $f_B(i) - it_{B^s}$.

In case (I) pattern (B.2), the maximum of $f_B(i) - it_{B^s}$ occurs when $jp \mod r = r - \gcd(p, q)$; *i.e.,* at the beginning of the maximal catch-up sequence. Therefore, Eq. (21) is also valid in this case.

To compute the minimum, we look only at the last firing of each batch of consecutive firings of B. As a lower bound, we assume that each sub-iteration is a catch-up sequence. Hence, $t^0_{B^\ell} = \min_j \left(t_B + \sigma_2 + \gamma_j \sigma'_1 + \sum_{i=1}^{j-1} (\sigma_2 + \gamma_i \sigma_1)\right)$.

So, $t^0_{B^\ell} = t_B + \min_j \left(\sigma_2 + \gamma_j \sigma'_1 + (j-1)t_A + \left\lfloor \frac{(j-1)p}{r} \right\rfloor \sigma_1\right)$. If $(j-1)p = ar + b$, then we have that $\gamma_j = \left\lfloor \frac{p+b-r}{r} \right\rfloor$ and $(j-1)t_A + \left\lfloor \frac{(j-1)p}{r} \right\rfloor \sigma_1 = \frac{bt_A}{p}$. Therefore, we rewrite the previous equation as $t^0_{B^\ell} = t_B + \sigma_2 - \sigma'_1 + \min\left(\left\lfloor \frac{p+b}{r} \right\rfloor \sigma'_1 + \frac{bt_A}{p}\right)$. Since $\left\lfloor \frac{p+b}{r} \right\rfloor$ can take only two values (either $\left\lfloor \frac{p}{r} \right\rfloor$ or $\left\lceil \frac{p}{r} \right\rceil$) and σ'_1 is negative, the minimum occurs either when $b = 0$ or for the minimum value of b (denoted b_0) that satisfies $\left\lfloor \frac{p+b}{r} \right\rfloor = \left\lceil \frac{p}{r} \right\rceil$.

So, $t^0_{B^\ell} = t_B + \sigma_2 - \sigma'_1 + \min\left\{\left\lfloor \frac{p}{r} \right\rfloor \sigma'_1, \left\lceil \frac{p}{r} \right\rceil \sigma'_1 + \frac{b_0 t_A}{p}\right\}$. But, $b_0 = r - (p \mod r)$. Therefore,

$$f_{B^\ell}(i) = \frac{qt_A}{p} i + t_B + \frac{p-r}{p} t_A + \left\lfloor \frac{p}{r} \right\rfloor \left(\frac{pt_B - qt_A}{p}\right) + \min\left\{\frac{qt_A - pt_B}{p}, \frac{r - (p \mod r)}{p} t_A\right\} \quad (54)$$

Actually, simulation shows that this lower bound is precise even when the lengths of catch-up sequences are longer than one.

pattern (B.3):

In case (I) pattern (B.3), the maximum of $f_B(i) - it_{B^s}$ occurs when $jr \mod (p-r) = \gcd(p,q)$; hence at the beginning of the maximal catch-up sequence. Therefore, Eq. (21) is also valid in this case.

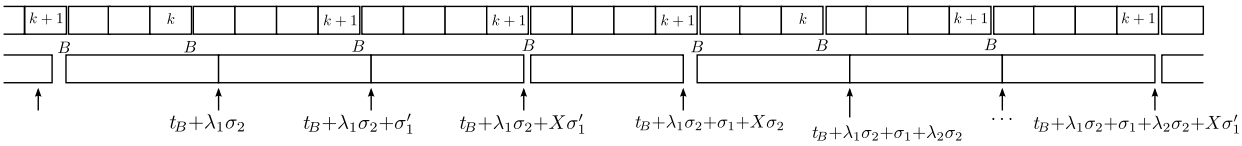


Figure 51: Case (II) pattern (B.3): values of $f_B(i) - it_{B^s}$.

To compute the minimum, we look only at the last firing of each batch of consecutive firings of B. Let us first consider case (II), illustrated in Fig. 51 where $\sigma_1' = t_B - t_{B^s} < 0$ and X is the smallest integer such that $r' + X(r' - t_A) \leq 0$ (*i.e.,* the overflow is caught up). So, $X = \left\lceil \frac{r'}{t_A - r'} \right\rceil$.

Since σ_1' is negative, we have that $t^0_{B^\ell} = \min\{t_B + X\sigma_1' - \sigma_1 + \min_j \left(j\sigma_1 + \sigma_2 \sum_{i=1}^{j} \lambda_i \right), t_B + \min_j \left(j\sigma_1 + \sigma_2 \sum_{i=1}^{j} \lambda_i \right) + \sigma_2\} = t_B + \min\{X\sigma_1' - \sigma_1, \sigma_2\}$. Therefore,

$$f_{B^\ell}(i) = \frac{qt_A}{p} i + t_B + \min\left\{ \frac{p-r}{r} t_A, \left\lceil \frac{r'}{t_A - r'} \right\rceil \left(\frac{pt_B - qt_A}{p} \right) + \frac{r}{p} t_A \right\} \tag{55}$$

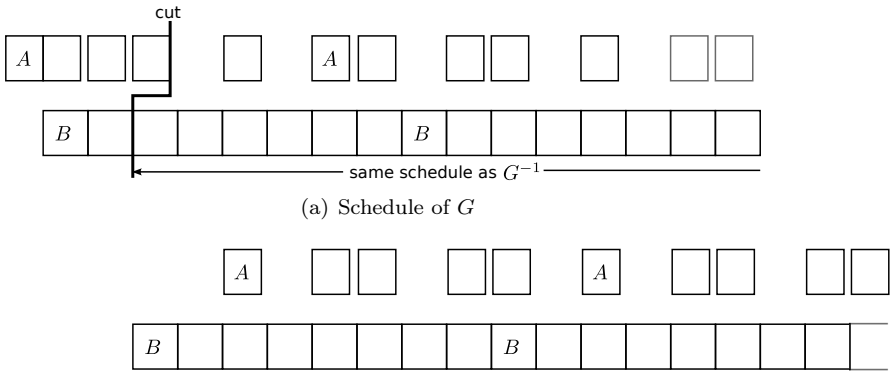
Actually, it is possible to show that this equation is equivalent to Eq. (54). According to that equation and after substituting $\lfloor \frac{p}{r} \rfloor$ by one and $p \mod r$ by $(p-r)$ (since $p < 2r$ in case B.3), we have that $t^0_{B^\ell} = t_B + \min\{\sigma_2, \sigma_1' - \sigma_1\}$. So, we need to prove that $\min\{\sigma_2, X\sigma_1' - \sigma_1\} = \min\{\sigma_2, \sigma_1' - \sigma_1\}$. Hence, we have to prove that $\sigma_2 \leq X\sigma_1' - \sigma_1 \Rightarrow \sigma_2 \leq \sigma_1' - \sigma_1$ and $\sigma_2 > X\sigma_1' - \sigma_1 \Rightarrow X = 1$. The first one is trivial since σ_1' is negative and $X \geq 1$. The second one is less evident. Suppose that $\sigma_2 > X\sigma_1' - \sigma_1$ when $X = 2$. So, $\sigma_2 > 2\sigma_1' - \sigma_1$ will imply that $t_A > 2r'$, and hence $X = \left\lceil \frac{r'}{t_A - r'} \right\rceil = 1$; *i.e.,* contradiction.

Let us now consider case (III). As a lower bound, we assume that each sub-iteration is a catch-up sequence. Therefore, the previous equation is a valid lower bound. Actually, simulation shows that this is a precise bound. □

C.2 Backward linearization

Proof. (**Property 5.1**) The key element to compute the backward lower bound linearization lies in the following observation, which relates backward linearization to forward linearization. Fig. 52(a) shows the ASAP schedule of graph $G = A \xrightarrow{8\ \ 5} B$ such that $t_A = 5$, $t_B = 6$ and the buffer size is equal to 22. Actor B imposes the highest load. Fig. 52(b) shows the ASAP schedule of the dual graph $G^{-1} = B \xrightarrow{5\ \ 8} A$ with the same buffer size. The producer B in G^{-1} imposes the highest load.

As illustrated in Fig. 52(a), there is a prologue 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 prologue can be a little 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.



(a) Schedule of G

(b) Schedule of G^{-1}

Figure 52: Relation between forward and backward linearizations for $G = A \xrightarrow{8 \quad 5} B$ with $t_A = 5$, $t_B = 6$ and $\theta_{A,B} = 22$.

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 prologue phase, the forward edge will contain d tokens while the backward edge will be empty; and this is exactly the dual graph G^{-1}. 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

$$i_0 p - j_0 q = d \tag{56}$$

This Diophantine equation is always solvable since the buffer size must be a multiple of $\gcd(p, q)$[5].

Let $\tilde{f}_A(i)$ denote the finish time of the ith firing of A in the ASAP schedule of the dual graph G^{-1}. Hence, we have

$$\forall i \geq 1.\ \tilde{f}_A(i) + s_B(j_0 + 1) \leq f_A(i + i_0) \tag{57}$$

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 firing of A in the dual graph, *i.e.,* $\tilde{f}_A(i)$, plus the shift due to the prologue phase, *i.e.,* $s_B(j_0 + 1)$.

As described in Section 5.1.2, 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 $\tilde{f}_A(i) \geq i t_{A^\ell} + \tilde{t}^0_{A^\ell}$. Eq. (58) is hence a valid lower bound linearization.

$$\forall i > i_0.\ f_A(i) \geq i t_{A^\ell} + t^0_{A^\ell} \tag{58}$$

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

Note that Eq. (58) is also a valid lower bound for all $i \leq i_0$. □

D Buffer Sizing for Acyclic graphs

D.1 Safe upper bounds

Proof. (**Property 6.1**) For a better understanding, we first present the proof for chains. Let G be the chain $\{A_1 \xrightarrow{p_1 \quad q_1} A_2 \xrightarrow{p_2 \quad q_2} A_3 \rightarrow \cdots \rightarrow A_n\}$, according to Eq. (5), the minimal period

[5] If the buffer size is not a multiple of $\gcd(p, q)$, then it can be diminished without affecting the ASAP schedule.

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)). 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.

To prove the general case of graphs without undirected cycles, it is sufficient to prove that there exists a schedule (not necessarily an ASAP schedule) of $G_=$ with period equals $\mathcal{P}_G$ and where the size of each buffer $A \xrightarrow{p \quad q} B$ is equal to $\theta^u_{A,B}$. In the computed schedule, each actor never gets idle once it starts.

The proof is based on the *Stretch* linearization method. For a graph $A \xrightarrow{p \quad q} B$ such that $z_A t_A = z_B t_B$, the upper bound linearization of B gives $\forall i.\ f_{B^u}(i) = t_B i + \frac{t_B}{q}(p + q - \gcd(p, q))$ (see Eq.(21)). If B starts at $s_B = \frac{t_B}{q}(p + q - \gcd(p, q))$, then a buffer with size equal to $\theta^u_{A,B}$ still allows the graph to achieve the maximal throughput. Indeed, the required buffer size is equal to (see Eq. (7)) $\max_j(jp - qx_j)$ such that x_j is the number of firings of B that finish before the start of the jth firing of A. Hence x_j is equal to the largest non-negative integer such that $(j-1)t_A \geq f_{B^u}(x_j)$. Therefore, $x_i = \left\lfloor \frac{(j-1)t_A - s_B}{t_B} \right\rfloor_0$ where $\lfloor x \rfloor_0 = \max(\lfloor x \rfloor, 0)$. Hence, the required size is

$$\max_j \left(jp - q \left\lfloor \frac{jp - (2p + q - \gcd(p, q))}{q} \right\rfloor_0 \right) = \theta^u_{A,B} \tag{59}$$

Using Eq. (21) transitively, we can compute s_{A_n}, the start time of actor A_n in a chain $A_1 \xrightarrow{p_1 \quad q_1} A_2 \to \cdots \xrightarrow{p_{n-1} \quad q_{n-1}} A_n$ (all actors impose the same load) after delaying each actor in the chain as indicated by the linearization method. It follows that

$$f_{A_n^u}(i) = t_{A_n} i + z_{A_n} t_{A_n} \sum_{i=1}^{n-1} \frac{p_i + q_i - \gcd(p_i, q_i)}{p_i z_{A_i}} \tag{60}$$

Note that a size $\theta^u_{A_i,A_{i+1}}$ for each channel in the chain will allow each actor to fire consecutively once it started. Hence, the execution will achieve the maximal throughput.

Suppose that there are two chains that start form the same root: chain $A_1 \to A_2 \to \ldots \to A_n$ and chain $A_1 \to A'_2 \to \ldots \to A'_n$. Delaying each actor as described above still allow to construct the desired schedule. This solves the problem of trees.

Suppose that there are two channels that enter a join node E: channel $C \xrightarrow{p \quad q} E$ and channel $D \xrightarrow{p' \quad q'} E$. We have $s_E = \max(s_1, s_2)$ with $s_1 = s_C + \frac{t_E}{q}(p + q - \gcd(p, q))$ and $s_2 = s_D + \frac{t_E}{q'}(p' + q' - \gcd(p', q'))$. Since the original graph G does not contain any undirected cycle, graphs $\downarrow C$ (obtained by restricting G to actor C and all its predecessors) and $\downarrow D$ are disjoint graphs.

If $s_2 > s_1$, then the schedule of graph $\downarrow D$ imposes a larger delay on the start time of actor E than graph $\downarrow C$. Since we want to compute a schedule where each actor never gets idle once it starts, we have to delay the start time of the schedule of the first graph $\downarrow C$ by $s_2 - s_1$ unit of time

in order to synchronize it with the schedule of the second graph. This will prevent the actors of the first graph from being blocked during the extra delay $s_2 - s_1$. By using this technique at each join node, we can compute the desired schedule. □

Proof. (**Property 6.2**) As in the proof of Property 6.1, we will construct a schedule of graph $G_=$ such that an actor never gets idle once it starts. Initially, the size of each channel $A \xrightarrow{p\ \ q} B$ is equal to $\theta^u_{A,B}$. For both chains, we construct a linear schedule as in the proof of Property 6.1. So, s_1 (resp. s_2) denotes the start time of the last actor A_n in the schedule of the first (resp. second) chain.

Since both schedules have the same source A_1 and same sink A_n, we have to synchronize both schedules such that the firings of A_1 and A_n in both schedules coincide. Hence, A_1 must start at time zero while A_n needs to start at time $\max(s_1, s_2)$.

If $s_2 > s_1$, the extra delay $(s_2 - s_1)$ imposed by the second chain will force actors of the first chain to block (if buffer sizes are not increased), which will finally decrease the throughput of the graph.

Let $A_{n-1} \xrightarrow{p\ \ q} A_n$ be the last channel in the first chain. Increasing the size of this channel by $\left\lceil \frac{s_2 - s_1}{t_{A_{n-1}}} \right\rceil p$ 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.2 Improving the upper bounds

Proof. (**Property 6.3**) Suppose that the chain follows the descending order: $\forall i.\ z_{A_i} t_{A_i} \geq z_{A_{i+1}} t_{A_{i+1}}$. The size θ_{A_1,A_2} allows actor A_1 to run consecutively, which is the same behavior as when there is no constraint on buffer sizes. Then, the size θ_{A_2,A_3} allows actor A_2 to fire as soon as it is enabled by actor A_1. Actually, if there were no dependence from A_1 to A_2, the size θ_{A_2,A_3} would allow actor A_2 to run consecutively. The same reasoning shows that all actors are fired as it there were no buffer size constraints. Therefore, the chain achieves its maximal throughput. The case of ascending order can be easily dealt with using the duality theorem. □

E Latency computation for acyclic graphs

E.1 Multi-iteration latency of acyclic graphs

Proof. (**Property 7.1**) This property follows immediately form the compositionality of the SDF-to-HSDF transformation, *i.e.,* $\mathit{HSDF}(G) = \bigcup_{g \in \mathcal{G}(G)} \mathit{HSDF}(g)$. Therefore, any maximal path in the DAG obtained by unfolding $\mathit{HSDF}(G)$ 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 $\mathit{HSDF}(G)$ unless both actors belong to the same chain. □

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399