INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Multi-Periodic Process Networks: Technical Report

Albert Cohen — Daniela Genius — Abdesselem Kortebi — Zbigniew Chamski — Marc Duranton — Paul Feautrier

N° 4496

July 2002

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--4496--FR+ENG

ROCQUENCOURT

Multi-Periodic Process Networks: Technical Report

Albert Cohen* , Daniela Genius*** , Abdesselem Kortebi** , Zbigniew Chamski** , Marc Duranton** , Paul Feautrier*

Thème 1 — Réseaux et systèmes
Projet A3

Rapport de recherche n° 4496 — July 2002 — 13 pages

Abstract: This paper aims at modeling video stream applications with structured data and multiple clocks. Multi-Periodic Process Networks (MPPN) are real-time process networks with an adaptable degree of synchronous behavior and a hierarchical structure. MPPN help to describe stream-processing applications and deduce resource requirements such as parallel functional units, throughput and buffer sizes.

Key-words: Process network, real time, stream processing, modeling video applications

* INRIA Rocquencourt - A3 Group
** Philips Research

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex (France)
Téléphone : +33 1 39 63 55 11 —Télécopie : +33 1 39 63 53 30

Réseaux de processus multi-périodiques

Résumé : Ce document décrit un modèle d'applications de traitement de flux vidéo avec des données structurées et des horloges différentes. Les Réseaux de Processus Multi-Périodiques (MPPN) constituent une classe de réseaux de processus structurés hiérarchiquement, exprimant des propriétés temps-réel et dont le degré de synchronisme est paramétrable. Les MPPN contribuent à la description d'applications de traitement de flux et permettent d'évaluer les ressources nécessaires, telles que le nombre d'unités fonctionnelles en parallèle, la bande passante et les tailles de buffers.

Mots-clés : Réseau de processus, temps réel, traitement de flux, modèle d'applications vidéo

Table of Contents

1 Context and Goals

The need arises for hardware units to handle new kinds of video applications, combining multiple streams, graphics and MPEG movies, leading to increased system complexity. When beginning the design of a video system, the engineer is primarily interested in quickly determining the hardware requirements to run an application under specific real-time constraints. For example, a 3D pipeline moves large amounts of data of different size and type: cutting-edge 3D games easily reach a texture bandwidth over $600\,MB/s$, Z-Buffer, rasterization of various triangle sizes over $160\,MB/s$, and intermediate buffers whose size depends on the algorithm, the architecture and the schedule.

Multi-Periodic Process Networks (MPPN) model heterogeneous video-stream applications and help resource allocation. They describe an application's structure and temporal behavior, not precise functionality of processes, and they may interact with a high-level language from which scheduling and resource allocation are determined. However, MPPN are *not* intended to model *reactive* systems with unpredictable input events [2] or *dynamic* process creation. On the opposite, our model provides precise information regarding the *steady state* of a *deterministic* application mapped to a parallel architecture. We believe MPPN are well suited to help the mapping of a video filter or 3D graphics pipeline to explicitly parallel micro-architectures, e.g. clustered VLIW embedded processors.

Section 2 discusses related work and Section 3 motivates the MPPN model with an introductory example and presents the network structure. The model is described in Section 4 and applied to performance evaluation in Section 5. Then, Section 6 describes a method to check and evaluate MPPN properties and Section 7 sketches useful extensions.

2 Related Work

Three theoretical models have influenced MPPN: Petri nets, data-flow graphs and Kahn Process Networks (KPN). Petri nets are inherently asynchronous and handle time constraints [3,12,7]. MPPN may be simulated by timed Petri nets but this does not bring precise schedule information. The sub-class of discrete event systems [1,4] enables scheduling and performance analysis but does not model token assembling/splitting. Data-flow graphs are a well-established means to describe asynchronous processing: various properties can be verified, such as bounded memory [10,5]. Both models capture repetitive actions through cyclic paths whose production rate compel performance, whereas stream-processing applications benefit from alternative descriptions such as lazy streams. Indeed, KPN [8] are closer to our approach: they provide (unbounded) FIFO buffers with blocking reads and non-blocking writes while enforcing deterministic control. But real-time is not considered and processes have no observable semantics. Synchronous approaches [6] are based on clock calculi and enable synchronous code generation, but static steady-state properties are not available. In our deterministic stream-processing context, properties such as degree of parallelism, buffer size and bandwidth are out of reach of these popular models. Other approaches are complementary to MPPN. Alpha [11] is a high-level language for semi-automatic scheduling and mapping of numerical applications to VHDL. Within the Ptolemy project [5], Compaan targets automatic KPN generation from MatLab loop nests [9]; it is also a powerful simulation tool.

KPN modeling, though frequently used in co-design, is insufficient for streams of structured data. As an introductory example consider *downscaling* of a video image is decomposed into a sequence of horizontal and vertical filtering. The former operates on pixels and the latter operates on lines — see Figure 1 for a simplified KPN model. A certain number of pixels/lines is used to determine the new, smaller number of pixels/lines. We assume a horizontal downscaling of 8:3 and a vertical downscaling of 9:4 (High Definition to Single Definition). Figure 1 describes the "data reordering" occurring within stripes, between the horizontal and the vertical filters. First of all, the hierarchy captures non-FIFO communication without resorting to an explicit reorder process. More importantly, each passage through a hierarchy boundary corresponds to

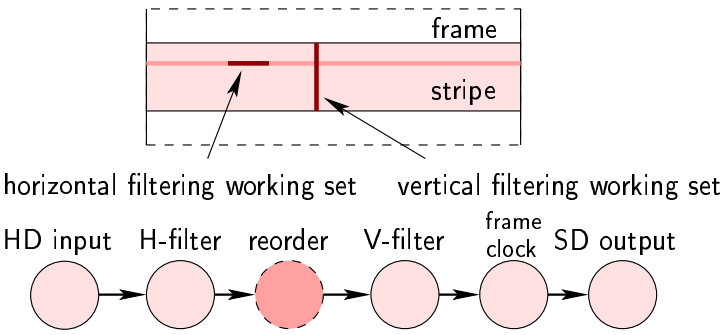

Fig. 1. Example: downscaler

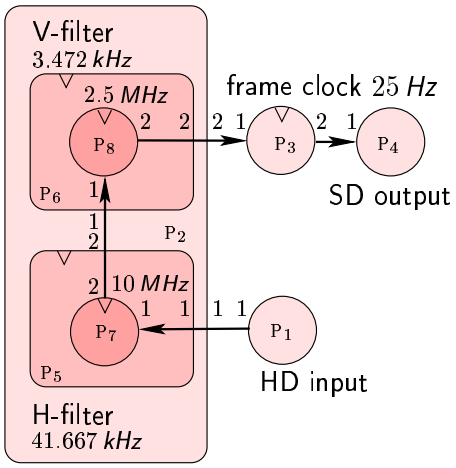

Fig. 2. Simple model of the downscaler

an explicit synchronization, where larger messages are considered, consisting of a fixed number of smaller messages. This hierarchical synchronization of events is called *multi-periodic*: it will be characterized through multiple, hierarchically layered, periodic schemes.

3 Network Structure

A Multi-Periodic Process Network (MPPN) is a 5-tuple $(\mathbf{P}, \sqsubseteq, \mathbf{C}, \text{IN}, \text{OUT})$, where $\mathbf{P}$ is a set of *processes*, $\sqsubseteq$ is a *hierarchical ordering* on $\mathbf{P}$ (its Hasse diagram is a forest), $\mathbf{C}$ is a set of *channels*, process $\text{P}_i \in \mathbf{P}$ is associated with *input ports* in $\text{IN}(\text{P}_i)$ and *output ports* in $\text{OUT}(\text{P}_i)$. P_i^j denotes port j of process P_i. Ordering $\sqsubseteq$ describes the hierarchy among processes: P_i is a *sub-process enclosed* by P_k if and only if $\text{P}_i \sqsubset \text{P}_k$. Moreover, a process P_i is *immediately* enclosed by P_k if and only if $\text{P}_i \sqsubset \text{P}_k$ and there is no other process enclosing P_i and enclosed by P_k. Processes which do not enclose other processes are called *atomic*; conversely, *compound* processes enclose of one or more sub-processes.

A channel connects process ports: P_i^jP_k^l represents a channel whose *source* is port P_i^j and whose *sink* is port P_k^l. Any port must belong to exactly one channel. Channels are defined inductively:

- if P_i and P_k are *immediately* enclosed by the same process or at the upper level, $j \in \text{OUT}(\text{P}_i)$, $l \in \text{IN}(\text{P}_k)$, then P_i^jP_k^l is a *flat atomic channel*;
- if P_i is *immediately* enclosed by P_k, $j \in \text{IN}(\text{P}_i)$, $l \in \text{IN}(\text{P}_k)$ (resp. $\text{OUT}(\text{P}_i)$, $\text{OUT}(\text{P}_k)$), then P_k^lP_i^j is a *downward atomic channel* (resp. P_i^jP_k^l is an *upward atomic channel*);
- if $\text{P}_i^j\text{P}_k^l \in \mathbf{C}$ and $\text{P}_k^l\text{P}_{k'}^{l'} \in \mathbf{C}$, then $\text{P}_i^j\text{P}_{k'}^{l'}$ is also a channel (not an atomic one); P_i^jP_k^l and P_k^lP_m^n are called *sub-channels* of P_i^jP_m^n.

$\mathbf{C}_{\text{FLAT}}$, $\mathbf{C}_{\text{DOWN}}$ and $\mathbf{C}_{\text{UP}}$ are the sets of flat, downward and upward atomic channels, respectively. The MPPN for the downscaler in Figure 2 illustrates these definitions. It is built from three compound processes, P_2, P_5, P_7, and five atomic ones. Digits accross process boundaries are port numbers: P_5^2P_6^1 and P_2^2P_3^1 are flat channels, P_5^1P_6^1 is a downward channel, P_6^2P_2^2 is an upward channel, etc.

A *path* π over the network is a list $\text{P}_{i_1}^{j_1}\text{P}_{i_2}^{j_2}\cdots\text{P}_{i_n}^{j_n}$ such that: $\text{P}_{i_m}^{j_m}\text{P}_{i_{m+1}}^{j_{m+1}}$ is either an *atomic* channel or a pair of input/output ports of the *same* process. E.g., any channel is a path, and $\text{P}_2^1\text{P}_5^1\text{P}_7^1\text{P}_7^2\text{P}_5^2\text{P}_6^1\text{P}_8^1\text{P}_8^2\text{P}_6^2\text{P}_2^2$ is a path in Figure 2. A port or process is *reachable* from another port or process if there exists a path from the latter to the former. Eventually, any output port of a compound process P_i must be reachable from an input port of P_i. For the sake of clarity, we only consider *acyclic* networks with *periodic* input streams (see Section 7 for extensions).

4 Network Semantics

We now enrich the network structure with data-flow activation and message semantics to model the execution of a stream-processing application.

During the course of execution, processes exchange messages and activate in response to receiving such messages. For each process P_i (resp. port P_i^j), the activation *count* is defined as the number of activations of P_i (resp. the number of messages hitting [1] port P_i^j) *since the last activation of the enclosing process* — or since the beginning of the execution if P_i is at the highest level of the hierarchy. Activation and message *dates* are defined likewise: date 0 corresponds to the last activation of the enclosing process — or the beginning of the execution of an outermost process — and all activation/message dates of sub-processes and ports are relative to this activation.

The model is designed such that local event dates only depend on the local event count, i.e., previous activations of the enclosing process have no "memory" effect. This locality property is one of the keys to compositionality — see Section 4.3. It also enables the following definition: an *execution* of a MPPN is a pair of non-decreasing functions $(\mathrm{ACT}, \mathrm{MSG})$, such that $\mathrm{ACT} : \mathbf{P} \times \mathbb{N} \to \mathbb{R}$ maps processes and activation counts to activation dates, $\mathrm{MSG} : \{\mathrm{P}_i^j \mid \mathrm{P}_i \in \mathbf{P} \wedge j \in \mathrm{IN}(\mathrm{P}_i) \cup \mathrm{OUT}(\mathrm{P}_i)\} \times \mathbb{N} \to \mathbb{R}$ maps process ports and message counts to message dates; $\mathrm{ACT}(\mathrm{P}_i, n)$ is the date of activation n of process P_i, $\mathrm{MSG}(\mathrm{P}_i^j, n)$ is the date of message n hitting port P_i^j.

4.1 Propagation in atomic channels

This section defines propagation equations for *atomic* channels only. When a big message is sent through a *flat* channel and decomposed into smaller ones, the first small message is received right after the big one is sent (pending some communication latency). Conversely, building a big message out of smaller ones takes additional time: many small messages must be sent before a big one is received. In both cases, n messages of size Q_k^l hitting port l of P_k through $\mathrm{P}_i^j\mathrm{P}_k^l$ correspond to $\lceil nQ_k^l/Q_i^j \rceil$ messages sent by P_i.

In addition, we assume a constant communication latency $c_{i,k}^{j,l}$ for an elementary message sent through an *atomic* channel $\mathrm{P}_i^j\mathrm{P}_k^l$. Recalling that $\mathrm{MSG}(\mathrm{P}_k^l, n)$ is the date of the $(n+1)$-th message hitting port l of process P_k,

$$\forall \mathrm{P}_i^j\mathrm{P}_k^l \in \mathbf{C}_{\mathrm{FLAT}} : \mathrm{MSG}(\mathrm{P}_k^l, n) = \mathrm{MSG}\big(\mathrm{P}_i^j, \lceil (n+1)Q_k^l/Q_i^j \rceil - 1\big) + c_{i,k}^{j,l}. \tag{1}$$

As a special case, when $Q_k^l \leq Q_i^j$, i.e., when decomposing messages into smaller ones, reception of message 0 at P_k^l coincides with reception of message n such that $\lceil (n+1)Q_k^l/Q_i^j \rceil = 1$, i.e., $0 \leq n \leq \lfloor Q_i^j/Q_k^l \rfloor - 1$:

$$\forall \mathrm{P}_i^j\mathrm{P}_k^l \in \mathbf{C}_{\mathrm{FLAT}}, 0 \leq n \leq \left\lfloor \frac{Q_i^j}{Q_k^l} \right\rfloor - 1 : \mathrm{MSG}(\mathrm{P}_k^l, n) = \mathrm{MSG}(\mathrm{P}_i^j, 0) + c_{i,k}^{j,l}. \tag{2}$$

Considering an upward channel, the propagation equation sums the activation date of the enclosing process and the local (relative) date of the last small message assembled to build an output message at port P_k^l:

$$\forall \mathrm{P}_i^j\mathrm{P}_k^l \in \mathbf{C}_{\mathrm{UP}} : \mathrm{MSG}(\mathrm{P}_k^l, n) = \mathrm{ACT}(\mathrm{P}_k, n) + \mathrm{MSG}\big(\mathrm{P}_i^j, \lceil Q_k^l/Q_i^j \rceil - 1\big). \tag{3}$$

Considering a downward channel, hierarchical composition enforces that *no message enters a compound process before it activates*. More precisely, when a message reaches an input port of a compound process P_i, this message is *not* propagated further on the channel (and possibly decomposed) before P_i activates on this very message. Activation of P_i coincides with the reception of a message at port P_k^l, since $Q_k^l \leq Q_i^j$ (decomposition into smaller messages). Therefore,

$$\forall \mathrm{P}_i^j\mathrm{P}_k^l \in \mathbf{C}_{\mathrm{DOWN}} : \mathrm{MSG}(\mathrm{P}_k^l, n) = 0. \tag{4}$$

[1] We say that a message *hits* an output port when it is sent through that port, and a message *hits* an input port when it is available for reception at this port.

4.2 Activation model

We consider a data-flow scheme: process activation starts as soon as there is at least one message on each input port. Let Q_i^j be the size of messages sent or received on port j of P_i. A process P_i enters activation n as soon as the following data-flow condition is met: every input port $j \in \mathrm{IN}(\mathrm{P}_i)$ has been hit by message n of size Q_i^j (except for special *clocked* processes, see Sections 4.4). The data-flow activation scheme is formalized as follows:

$$\mathrm{ACT}(\mathrm{P}_i, n) = \max_{j \in \mathrm{IN}(\mathrm{P}_i)} \mathrm{MSG}(\mathrm{P}_i^j, n). \tag{5}$$

This definition allows multiple overlapping activations of a process.

Considering an *atomic* process P_i, we call ℓ_i^j the latency of P_i for sending a message through output port j. It is defined as the elapsed time between an activation of P_i and the corresponding output of a message through port j, supposed *constant* for all executions of the process:

$$\mathrm{MSG}(\mathrm{P}_i^j, n) = \mathrm{ACT}(\mathrm{P}_i, n) + \ell_i^j. \tag{6}$$

This constant latency will be extended to *compound* processes in Section 4.3.

While any size change of messages is possible when traversing a process or channel, the same is not true when traversing a path. As a consequence of the previous equations, in order to ensure compositionality, message sizes must obey a strict *scaling rule* when traversing hierarchy boundaries. Let us consider a *compound* process P_i, an input port $j \in \mathrm{IN}(\mathrm{P}_i)$ and an output port $l \in \mathrm{OUT}(\mathrm{P}_i)$. If these ports are connected through a path π of channels and sub-processes of P_i, one single message occurring at P_i^j may traverse several decomposition/assembling stages through π, but it must yield one single message occurring at P_i^l. Let us define the *floor-product* over a list of scalars as the iterated product and floor operations:

$$\left\lfloor \prod \right\rfloor_{1 \le m \le n} x_m = \begin{cases} \lfloor x_1 \rfloor \text{ if } n = 1 \\ \left\lfloor x_n \times \left\lfloor \prod \right\rfloor_{1 \le m \le n-1} x_m \right\rfloor \text{ otherwise } . \end{cases}$$

The latter property of compound processes is ensured when the *product* of all factors Q_i^j / Q_k^l over each channel $\mathrm{P}_i^j \mathrm{P}_k^l \in \pi$ is equal to 1, and when the *floor-product* on a prefix of π is always greater than or equal to 1:

$$\prod_{\mathrm{P}_i^j \mathrm{P}_k^l \in \pi} \frac{Q_i^j}{Q_k^l} = 1 \tag{7}$$

$$\forall \pi' \text{ prefix of } \pi : \left\lfloor \prod \right\rfloor_{\mathrm{P}_i^j \mathrm{P}_k^l \in \pi'} \frac{Q_i^j}{Q_k^l} \ge 1. \tag{8}$$

Notice that, when traversing an upward or downward channel, (8) forbids messages at the upper level to be smaller than messages at the lower one.

Eventually, activation of a *source* (input-less) process P_i is not constrained by any data-flow scheme. We assume a *periodic behavior* instead: considering two real numbers $\mathrm{ACT}(\mathrm{P}_i, 0)$ — the *reference date* — and $\mathrm{PER}(\mathrm{P}_i)$ — the *period*,

$$\mathrm{ACT}(\mathrm{P}_i, n) = \mathrm{ACT}(\mathrm{P}_i, 0) + n\mathrm{PER}(\mathrm{P}_i). \tag{9}$$

4.3 Latencies of compound processes

Consider an output port j of a compound process P_i. We can prove that every activation of P_i sends one message through P_i^j after a *constant* latency. This result lies in the data-flow equations (message propagation and activation rules are time invariant) and in the scale factor constraint (7) which ensures that any single

activation of P_i sends exactly one message through P_i^j. We may thus extend (6) to *compound* processes: $\mathrm{MSG}(\mathrm{P}_i^j, n) - \mathrm{ACT}(\mathrm{P}_i, n)$ is a constant, ℓ_i^j can be computed for $n = 0$ based on information at the lower level:

$$\forall \mathrm{P}_i \in \mathbf{P}, j \in \mathrm{OUT}(\mathrm{P}_i) : \ell_i^j = \mathrm{MSG}(\mathrm{P}_i^j, 0) - \mathrm{ACT}(\mathrm{P}_i, 0). \tag{10}$$

Thus, MPPN exhibit compositional semantics. Latencies of compound processes do not depend on the surrounding network: they are computed once and for all.

4.4 Clocked processes

We provide an extended kind of process to synchronize streams over a fixed clock period: *clocked* processes. These processes can be either atomic or compound, and their activation rule is generalized from ordinary processes. Considering a clocked process P_i, an *internal clock* starts at the reference date $\mathrm{ACT}(\mathrm{P}_i, 0)$, and subsequent activations may only occur *one at a time* when all input messages are present and when an *internal clock tick* occurs. To put it simple, a clocked process has a double role of (local) *sampling* and *delay*.

Formally, a clocked process P_i is characterized through a *clock frequency* f_i; calling $\alpha(n)$ the date given by (5) when interpreting P_i as an ordinary process,

$$\mathrm{ACT}(\mathrm{P}_i, n) = \max\left(\frac{\lceil(\alpha(n) - \mathrm{ACT}(\mathrm{P}_i, 0))fi\rceil}{f_i} + \mathrm{ACT}(\mathrm{P}_i, 0), \mathrm{ACT}(\mathrm{P}_i, n-1) + \frac{1}{f_i}\right). \tag{11}$$

When enclosed in compound processes, MPPN clocks behave differently from hardware clocks (whose semantics is exclusive): two independent activations of a compound process may trigger overlapping streams of events on clocked sub-processes. This is required to preserve compositionality. In practice, the designer may want the enclosing process to be clocked itself so that executions of the clocked sub-process are sequential, e.g., in dividing the frequency of the clocked sub-process by the hierarchical scale factor.

5 High Level Properties

From the abovementioned MPPN semantics, one may deduce resource requirements of the application. In this paper, we focus on conservative estimates for the number of functional units, bandwidth and buffers. We derive global (absolute) properties from the product of local evaluations.

5.1 Asymptotic Periodic Execution

We will now prove that every process follows a steady-state scheme which extends and relaxes the periodic constraint (9) on source processes. We define an *average period* $\mathrm{PER}(\mathrm{P}_i)$, a *burstiness* $\mathrm{ADV}(\mathrm{P}_i)$ for each process P_i, and *burstinesses* $\mathrm{ADV}(\mathrm{P}_i^j)$ for each port P_i^j, such that

$$\forall n \geq 0 : (n - \mathrm{ADV}(\mathrm{P}_i))\mathrm{PER}(\mathrm{P}_i) \leq \mathrm{ACT}(\mathrm{P}_i, n) - \mathrm{ACT}(\mathrm{P}_i, 0) \leq n\mathrm{PER}(\mathrm{P}_i) \tag{12}$$

$$\forall n \geq 0 : (n - \mathrm{ADV}(\mathrm{P}_i^j))\mathrm{PER}(\mathrm{P}_i) \leq \mathrm{MSG}(\mathrm{P}_i^j, n) - \mathrm{MSG}(\mathrm{P}_i^j, 0) \leq n\mathrm{PER}(\mathrm{P}_i). \tag{13}$$

Intuitively, the burstiness $\mathrm{ADV}(\mathrm{P}_i)$ is the maximal number of *advance* activations of P_i, i.e., the maximal number of activations ahead of the periodic execution scheme. This parameter encompasses both *deterministic bursts* of early messages and *jittering streams* with earliest/latest bounds (and possibly, periodic resynchronization). Even under the worst-case conditions, better evaluations of $\mathrm{ACT}(\mathrm{P}_i, n)$ (often exact ones) can be hoped for: deterministic event bursts can be characterized effectively within the relaxed periodic scheme.

Let us now prove (12) and (13), by induction on (6), (1) and (5). Of course, (9) is a special case of (12) for source processes.

Message propagation Suppose (12) holds for a process P_i. Messages occur at an output port P_i^j after a constant delay, hence output message burstiness is equal to activation burstiness:

$$\forall j \in \mathrm{OUT}(\mathrm{P}_i) : \mathrm{ADV}(\mathrm{P}_i^j) = \mathrm{ADV}(\mathrm{P}_i). \tag{14}$$

This equation stands for any process, atomic or compound, and of course for any *upward* channel.

Consider any atomic channel $\mathrm{P}_i^j\mathrm{P}_k^l$, and suppose (13) holds in P_i^j. Sending one message and $\mathrm{ADV}(\mathrm{P}_i^j)$ advance messages at port j of P_i corresponds to sending $Q_i^j(1 + \mathrm{ADV}(\mathrm{P}_i^j))$ units of data. These data are received as one message plus $\mathrm{ADV}(\mathrm{P}_k^l)$ advance messages at port l of a process P_k, i.e., $Q_k^l(1 + \mathrm{ADV}(\mathrm{P}_k^l))$ units. For *flat* channels, the result is the following:

$$\forall \mathrm{P}_i^j\mathrm{P}_k^l \in \mathbf{C}_{\mathrm{FLAT}} : Q_k^l(1 + \mathrm{ADV}(\mathrm{P}_k^l)) = Q_i^j(1 + \mathrm{ADV}(\mathrm{P}_i^j)); \tag{15}$$

for *downward* channels, a single activation of the enclosing process is considered:

$$\forall \mathrm{P}_i^j\mathrm{P}_k^l \in \mathbf{C}_{\mathrm{DOWN}} : Q_k^l(1 + \mathrm{ADV}(\mathrm{P}_k^l)) = Q_i^j. \tag{16}$$

Notice that (15) and (16) may require burstinesses to be non-integer. In addition, communication may be implemented through *bounded buffers* as long as asymptotic data throughput is the same at both ends of a channel:

$$\forall \mathrm{P}_i^j\mathrm{P}_k^l \in \mathbf{C} : \frac{Q_k^l}{\mathrm{PER}(\mathrm{P}_k)} = \frac{Q_i^j}{\mathrm{PER}(\mathrm{P}_i)}. \tag{17}$$

Every-time a splitter or selector P_i (resp. P_k) is considered in the previous equations, one must multiply Q_i^j (resp. Q_k^l) by $s_i^j(s_i)/s_i$ (resp. $s_k^l(s_k)/s_k$).

Activation Activation burstiness can be deduced from the data-flow scheme characterized by (5), replacing $\mathrm{ACT}(\mathrm{P}_i, n)$ and $\mathrm{MSG}(\mathrm{P}_i^j, n)$ by their lower bounds. The result is that processes tend to "smooth" message bursts and initialization delays:

$$\mathrm{ADV}(\mathrm{P}_i) = \min_{j \in \mathrm{IN}(\mathrm{P}_i)} \left\{ \mathrm{ADV}(\mathrm{P}_i^j) + \frac{\mathrm{ACT}(\mathrm{P}_i, 0) - \mathrm{MSG}(\mathrm{P}_i^j, 0)}{\mathrm{PER}(\mathrm{P}_i)} \right\}. \tag{18}$$

This is a two-phase computation: on a given stream, sum up the burstiness and the number of messages that precede activation 0, then minimize these adjusted burstinesses. There are two simple cases: when initial activation and message dates coincide, $\mathrm{ADV}(\mathrm{P}_i) = \min_{j \in \mathrm{IN}(\mathrm{P}_i)} \mathrm{ADV}(\mathrm{P}_i^j)$ only depends on burstinesses of input message streams; and when input message burstinesses are equal to some value v, $\mathrm{ADV}(\mathrm{P}_i)$ is also equal to v.

Clocked processes Since the activation model differs from usual processes, burstiness of clocked processes obeys a specific rule. Consider a clocked process P_i. Enforcing $\mathrm{ACT}(\mathrm{P}_i, n) \geq \mathrm{ACT}(\mathrm{P}_i, n-1) + 1/f_i$ requires that $\mathrm{ACT}(\mathrm{P}_i, n) - \mathrm{ACT}(\mathrm{P}_i, n-d) \geq \mathrm{ACT}(\mathrm{P}_i) + d/f_i$. The lower bound on $\mathrm{ACT}(\mathrm{P}_i, n) - \mathrm{ACT}(\mathrm{P}_i, n-d)$ is reached when considering the lower bound on $\mathrm{ACT}(\mathrm{P}_i, n)$ and the upper bound on $\mathrm{ACT}(\mathrm{P}_i, n-d)$. Therefore, for a given value of d,

$$(d - \mathrm{ADV}(\mathrm{P}_i))\mathrm{PER}(\mathrm{P}_i) + \mathrm{ACT}(\mathrm{P}_i, 0) \geq \mathrm{ACT}(\mathrm{P}_i, 0) + \frac{n}{f_i},$$

hence

$$\mathrm{ADV}(\mathrm{P}_i) \leq d\Big(1 - \frac{1}{\mathrm{PER}(\mathrm{P}_i) f_i}\Big).$$

Calling ν_i the burstiness calculated for P_i without taking the clock into account, the maximal number of advance messages, d, is equal to ν_i, thus

$$\mathrm{ADV}(\mathrm{P}_i) \leq \nu_i\Big(1 - \frac{1}{\mathrm{PER}(\mathrm{P}_i) f_i}\Big).$$

(Notice $1 - \frac{1}{f_i \mathrm{PER}(\mathrm{P}_i)}$ is non-negative.) Eventually, we get a safe evaluation of the burstiness for a clocked process P_i:

$$\mathrm{ADV}(\mathrm{P}_i) = \Big(1 - \frac{1}{f_i \mathrm{PER}(\mathrm{P}_i)}\Big) \min_{j \in \mathrm{IN}(\mathrm{P}_i)} \Big\{ \mathrm{ADV}(\mathrm{P}_i^j) + \frac{\mathrm{ACT}(\mathrm{P}_i, 0) - \mathrm{MSG}(\mathrm{P}_i^j, 0)}{\mathrm{PER}(\mathrm{P}_i)} \Big\}. \tag{19}$$

As a corollary, it is mandatory that $f_i \geq 1/\mathrm{PER}(\mathrm{P}_i)$. When $\mathrm{PER}(\mathrm{P}_i) = 1/f_i$, (19) expectedly shows that $\mathrm{ADV}(\mathrm{P}_i) = 0$, enabling stream resynchronization at a fixed periodic scheme (e.g., for video output). Conversely, an infinite frequency makes the clock constraint redundant with the non-decreasing ordering of activations, leading to a normal process. In the general case, the clock provides designer control on the upper bound of activation burstiness.

5.2 Global Properties

We call ℓ_i the latency for P_i to complete an execution, i.e., the maximum of ℓ_i^j at output ports j of P_i: $\ell_i = \max_{j \in \mathrm{OUT}(\mathrm{P}_i)} \ell_i^j$. Let $\mathsf{overlap}(\mathrm{P}_i, d)$ denote the maximum number of executions of P_i during a given period of time d, and triggered by a single activation of the enclosing process (if P_i is a sub-process):

$$\mathsf{overlap}(\mathrm{P}_i, d) = \min\big(\lceil f_i d \rceil, \lceil d/\mathrm{PER}(\mathrm{P}_i) + \mathrm{ADV}(\mathrm{P}_i) \rceil\big).$$

We proved that the (absolute) maximal number of parallel executions of a process P_i is bounded by the product of the local maximal number of activations of P_i and all its enclosing processes during the same duration ℓ_i:

$$\mathsf{maxpll}(\mathrm{P}_i) = \prod_{\mathrm{P}_i \sqsubseteq \mathrm{P}_k} \mathsf{overlap}(\mathrm{P}_k, \ell_i).$$

5.3 Port and Channel Bandwidth

Depending on the architecture and the resource allocation strategy, ports associated with physical input/output may be identified. On this subset, it is legitimate to ask for an estimate of the average and maximal bandwidths. Such estimates can be built from the periods, burstinesses and overlapping factors. Our current model assumes that actual loads/stores are distributed evenly over the whole access period ℓ_i^j. This hypothesis is optimistic, but finer evaluations can be crafted following the same reasoning.

Port bandwidth is of critical interest when implementing process communications through shared-memory buffers. From (17), the average (resp. maximal) traffic associated with a port P_i^j mapped to a communication device (bus, shared memory, network, etc.) is denoted by $\mathsf{avgbw}(\mathrm{P}_i^j)$ (resp. $\mathsf{maxbw}(\mathrm{P}_i^j)$):

$$\mathsf{avgbw}(\mathrm{P}_i^j) = \frac{Q_i^j}{\mathrm{PER}(\mathrm{P}_i)} \tag{20}$$

$$\mathsf{maxbw}(\mathrm{P}_i^j) = Q_i^j \Big(\frac{1}{\mathrm{PER}(\mathrm{P}_i)} + \frac{\mathrm{ADV}(\mathrm{P}_i)}{\ell_i} \prod_{\mathrm{P}_i \sqsubset \mathrm{P}_k} \mathsf{overlap}(\mathrm{P}_k, \ell_i) \Big). \tag{21}$$

Indeed, in the context of the process immediately enclosing P_i (if any), periodic activation of P_i generates a throughput $Q_i^j/\mathrm{PER}(\mathrm{P}_i)$ over $\mathrm{P}_i^j \mathrm{P}_k^l$, and at most $\mathrm{ADV}(\mathrm{P}_i)$ advance messages may overlap during message generation on port P_i^j.

For a clocked process P_i, the upper bound is

$$\mathsf{maxbw}(\mathrm{P}_i^j) = Q_i^j \min\Big(f_i \prod_{\mathrm{P}_i \sqsubset \mathrm{P}_k} \mathsf{overlap}(\mathrm{P}_k, \ell_i), \frac{1}{\mathrm{PER}(\mathrm{P}_i)} + \frac{\mathrm{ADV}(\mathrm{P}_i)}{\ell_i} \prod_{\mathrm{P}_i \sqsubset \mathrm{P}_k} \mathsf{overlap}(\mathrm{P}_k, \ell_i) \Big). \tag{22}$$

Channel bandwidth provides some insight about network contention when focusing on distributed architectures. If "sufficiently large" buffers are available at both ends of the communication device, the channel bandwidth (average or maximal) is the minimum of both port bandwidths. Depending on the communication protocol and buffers, other estimates might be more appropriate.

5.4 Input and Output Buffers

We describe a method to bound buffer size for any *atomic* channel, not considering architecture-specific buffer requirements. The minimal size of a buffer for channel $\mathrm{P}_i^j\mathrm{P}_k^l$ is the maximum amount of temporary data that must be stored during message propagation through this channel; it is denoted by $\mathsf{maxbuf}(\mathrm{P}_i^j\mathrm{P}_k^l)$. Such a buffer must hold all the messages sent to the channel by P_i and not yet received by P_k, i.e., the difference between data sent and received. An upper bound of this difference is evaluated from the liveness of a message and the product of overlapping factors.

For a flat channel, the maximal liveness of a message must cope with the input/output delays for reading/writing the message. Indeed, propagation rules in atomic channels assume zero-latency read/write operations. A conservative approximation of this latency is obtained in considering that message output/input lasts during the whole activation of the input/output process. This is equivalent to replacing a message output date by an activation date of the sending process, and a message input date by an activation date of the receiving process plus the process latency:[2]

$$\max_{n\geq 0}\Big(\mathrm{ACT}\Big(\mathrm{P}_k,\Big\lceil(n+1)\frac{Q_i^j}{Q_k^l}\Big\rceil-1\Big)+\ell_k-\mathrm{ACT}(\mathrm{P}_i,n)\Big).$$

When $\mathrm{P}_i^j\mathrm{P}_k^l$ is enclosed in a compound process P_g, the scaling rule (7) enforces that one activation of P_g triggers exactly $\lfloor \mathrm{PER}_g/\mathrm{PER}_i\rfloor$ activations of P_i; indeed, all these activations are triggered by one single entering message on each input port of P_i. One may thus restrict the computation of the maximum to $0\leq n<\lfloor \mathrm{PER}_g/\mathrm{PER}_i\rfloor$. We can deduce the buffer size for an atomic flat channel enclosed in a compound process:

$$\forall \mathrm{P}_i^j\mathrm{P}_k^l\in\mathbf{C}_{\mathrm{FLAT}},\mathrm{P}_i\sqsubset\mathrm{P}_g:\mathsf{maxbuf}(\mathrm{P}_i^j\mathrm{P}_k^l)=$$
$$Q_i^j\prod_{\mathrm{P}_i\sqsubseteq\mathrm{P}_{i'}\sqsubset\top}\mathsf{overlap}\Big(\mathrm{P}_{i'},\max_{0\leq n<\left\lfloor\frac{\mathrm{PER}_g}{\mathrm{PER}_i}\right\rfloor}\Big(\mathrm{ACT}\Big(\mathrm{P}_k,\Big\lceil(n+1)\frac{Q_i^j}{Q_k^l}\Big\rceil-1\Big)+\ell_k-\mathrm{ACT}(\mathrm{P}_i,n)\Big)\Big)\quad(23)$$

Conversely, if P_i is at the highest level of the hierarchy, one may only guarantee that the maximal liveness is bounded by

$$\Big\lceil\frac{Q_i^j}{Q_k^l}\Big\rceil\mathrm{PER}(\mathrm{P}_k)+\mathrm{ACT}(\mathrm{P}_k,0)+\ell_k+\mathrm{ADV}(\mathrm{P}_i)\mathrm{PER}(\mathrm{P}_i)-\mathrm{ACT}(\mathrm{P}_i,0).$$

This bound enables a conservative approximation of the buffer size for an atomic flat channel at the highest level of the hierarchy:

$$\forall \mathrm{P}_i^j\mathrm{P}_k^l\in\mathbf{C}_{\mathrm{FLAT}},\mathrm{P}_i\sqsubset\top:\mathsf{maxbuf}(\mathrm{P}_i^j\mathrm{P}_k^l)=$$
$$Q_i^j\prod_{\mathrm{P}_i\sqsubseteq\mathrm{P}_{i'}\sqsubset\top}\mathsf{overlap}\Big(\mathrm{P}_{i'},\Big\lceil\frac{Q_i^j}{Q_k^l}\Big\rceil\mathrm{PER}(\mathrm{P}_k)+\mathrm{ACT}(\mathrm{P}_k,0)+\ell_k+\mathrm{ADV}(\mathrm{P}_i)\mathrm{PER}(\mathrm{P}_i)-\mathrm{ACT}(\mathrm{P}_i,0)\Big)\quad(24)$$

For upward and downward channels, liveness can be computed directly from the reference dates:

$$\forall \mathrm{P}_i^j\mathrm{P}_k^l\in\mathbf{C}_{\mathrm{UP}}:\mathsf{maxbuf}(\mathrm{P}_i^j\mathrm{P}_k^l)=Q_i^j\prod_{\mathrm{P}_i\sqsubseteq\mathrm{P}_{i'}\sqsubset\top}\mathsf{overlap}\big(\mathrm{P}_{i'},\ell_k^l-\mathrm{MSG}(\mathrm{P}_i^j,0)\big)\quad(25)$$

$$\forall \mathrm{P}_i^j\mathrm{P}_k^l\in\mathbf{C}_{\mathrm{DOWN}}:\mathsf{maxbuf}(\mathrm{P}_i^j\mathrm{P}_k^l)=Q_i^j\prod_{\mathrm{P}_i\sqsubseteq\mathrm{P}_{i'}\sqsubset\top}\mathsf{overlap}\big(\mathrm{P}_{i'},\mathrm{ACT}(\mathrm{P}_k,0)\big).\quad(26)$$

6 Network Analysis

The analysis of a multi-periodic process network consists in solving the above equations. Let us sketch an algorithm for MPPN analysis and verification.

[2] Finer evaluations and communication models are under way; they will be exposed in future versions of this work.

Input. A multi-periodic process network, Q_i^j for each port, $c_{i,k}^{j,l}$ for each atomic flat channel, ℓ_i^j for each atomic process, reference date $\text{ACT}(\text{P}_i, 0)$ for each source process (e.g., 0), period $\text{PER}(\text{P}_i)$ for one process per weakly-connected component of the network, burstiness $\text{ADV}(\text{P}_i)$ for each source process. Optional values for other parameters, e.g., burstinesses and periods at sink processes.

Output. Values for all parameters or contradiction.

Resolution. The algorithm is decomposed into four phases.

1. Perform a topological sort of the network. Check the scaling rule of all compound processes, using (7).
2. Compute $\text{PER}(\text{P}_i)$ traversing the network incrementally, starting from processes with known periods using (17) and checking for consistency.
3. Traverse the hierarchical structure bottom-up, applying the following steps:
   - choose a compound process P_i whose sub-processes have known latencies;
   - compute (relative) reference dates for sub-processes, using (1), (5) and (6);
   - deduce latency ℓ_i^j for every output port j, using (10).
4. Compute $\text{ADV}(\text{P}_i)$ and $\text{ADV}(\text{P}_i^j)$ through a top-down traversal, following the topological ordering at each hierarchical level, using (14), (15) and (18).

From the output of this algorithm, one may deduce the degree of parallelism, bandwidths and buffer sizes for all processes, ports and channels. We may now show the results on the introductory example.

Input. We consider a pixel unit for all message sizes, $Q_1^1 = Q_2^1 = 1920 \times 1080$, $Q_2^2 = Q_3^1 = Q_3^2 = Q_4^1 = 720 \times 480$, $Q_7^1 = 8$, $Q_5^1 = 1920$, $Q_7^2 = 3$, $Q_5^2 = 720$, $Q_8^1 = 9$, $Q_6^1 = 720 \times 9$, $Q_8^2 = 4$, $Q_6^2 = 720 \times 4$. Some latencies, activation dates, burstinesses, and periods are already given: $\text{ACT}(\text{P}_1, 0) = 0$, $\text{ADV}(\text{P}_1) = 0$, $\text{PER}(\text{P}_4) = 40\,ms$ $(25\,Hz)$, $\text{ADV}(\text{P}_4) = 0$, $\ell_1^1 = 0$ (source process), $\ell_3^2 = 1\,ms$, $\ell_7^2 = 200\,ns$, $\ell_8^2 = 1\,\mu s$, $c_{1,2}^{1,1} = c_{2,3}^{2,1} = c_{3,4}^{2,1} = 1\,\mu s$ (communication), $c_{5,6}^{2,1} = 100\,ns$ (local buffer). Clocks must be such that $f_i \geq 1/\text{PER}(\text{P}_i)$: we choose $f_7 = 10\,MHz$ and $f_8 = 2.5\,MHz$, and we deduce clocks for P_5 and P_6 from the hierarchical scale factors: $f_5 = f_7 \times (3/720) = 41.67\,kHz$, $f_6 = f_8 \times (4/(4 \times 720)) = 3.47\,kHz$.

Resolution. 1. Topological ordering: P_1, P_2, P_5, P_7, P_6, P_8, P_3, P_4. Scale factors for P_5: $(8/1920) \times (720/3) = 1$; for P_6: $(9/1080) \times (480/4) = 1$; for P_2: $(1920/(1920 \times 1080)) \times (720 \times 9/720) \times (720 \times 480/(720 \times 4) = 1$.

2. Process P_5: (1) and (5) give $\text{MSG}(\text{P}_7^1, 0) = \text{ACT}(\text{P}_7, 0) = 0$, then (6) gives $\text{MSG}(\text{P}_7^2, 0) = 200\,ns$, (1) gives $\text{MSG}(\text{P}_5^2, 0) = \text{MSG}(\text{P}_7^2, 239)$, (6) gives $\text{MSG}(\text{P}_5^2, 0) = \text{ACT}(\text{P}_7, 239) + 200\,ns$, (11) gives $\text{MSG}(\text{P}_5^2, 0) = \text{ACT}(\text{P}_7, 0) + 239 \times 100\,ns + 200\,ns = 24.1\,\mu s$. Finally, (10) gives $\ell_5^2 = 24.1\,\mu s$.
   Process P_6: (1) and (5) give $\text{MSG}(\text{P}_8^1, 0) = \text{ACT}(\text{P}_8, 0) = 0$, then (6) gives $\text{MSG}(\text{P}_8^2, 0) = 1\,\mu s$, (1) gives $\text{MSG}(\text{P}_6^2, 0) = \text{MSG}(\text{P}_8^2, 719)$, (6) gives $\text{MSG}(\text{P}_6^2, 0) = \text{ACT}(\text{P}_8, 719) + 1\,\mu s$, (11) gives $\text{MSG}(\text{P}_6^2, 0) = \text{ACT}(\text{P}_8, 0) + 719 \times 400\,ns + 1\,\mu s = 288.6\,\mu s$. Finally, (10) gives $\ell_6^2 = 288.6\,\mu s$.
   Process P_2: (1) and (5) give $\text{MSG}(\text{P}_5^1, 0) = \text{ACT}(\text{P}_5, 0) = 0$, (6) gives $\text{MSG}(\text{P}_5^2, 0) = 24.1\,\mu s$, (1) gives $\text{MSG}(\text{P}_6^1, 0) = \text{MSG}(\text{P}_5^2, 7) + 100\,ns$, (6) gives $\text{MSG}(\text{P}_6^1, 0) = \text{ACT}(\text{P}_5, 7) + 24.1\,\mu s + 100\,ns$, (11) gives $\text{MSG}(\text{P}_6^1, 0) = 7/41.667\,ms + 24.2\,\mu s = 192.2\,\mu s$. (5) gives $\text{ACT}(\text{P}_6, 0) = 192.2\,\mu s$, (6) gives $\text{MSG}(\text{P}_6^2, 0) = 480.8\mu s$, (1) gives $\text{MSG}(\text{P}_2^2, 0) = \text{MSG}(\text{P}_6^2, 119)$, (6) gives $\text{MSG}(\text{P}_2^2, 0) = \text{ACT}(\text{P}_6, 119) + 288.6\,\mu s$, (11) gives $\text{MSG}(\text{P}_2^2, 0) = 192.2\,\mu s + \max(\lceil 3472 \times 24 \times 10^{-6} \times 9 \times 119 \rceil / 3.472, 119/3.472)\,ms + 288.6\,\mu s = 288 \times 119\,\mu s + 480.8\,\mu s = 34.752\,ms$, and (10) gives $\ell_2^2 = 34.752\,ms$.
   Highest level: (6) gives $\text{MSG}(\text{P}_1^1, 0) = 0$, (1) gives $\text{MSG}(\text{P}_2^1, 0) = 1\,\mu s$, (5) gives $\text{ACT}(\text{P}_2, 0) = 1\,\mu s$, (6) gives $\text{MSG}(\text{P}_2^2, 0) = 34.752\,ms$, (1) gives $\text{MSG}(\text{P}_3^1, 0) = 34.753\,ms$, (5) gives $\text{ACT}(\text{P}_3, 0) = 34.753\,ms$, (6) gives $\text{MSG}(\text{P}_3^2, 0) = 35.753\,ms$, and (5) gives $\text{ACT}(\text{P}_4, 0) = 35.754\,ms$.

3. Highest level: (14) gives $\text{ADV}(\text{P}_1^1) = 0$, (15) gives $\text{ADV}(\text{P}_2^1) = 0$, (18) gives $\text{ADV}(\text{P}_2) = 1$, (14) gives $\text{ADV}(\text{P}_2^2) = 0$, (15) gives $\text{ADV}(\text{P}_3^1) = 0$, (19) gives $\text{ADV}(\text{P}_3) = 0$ because $\text{PER}(\text{P}_3) = 1/f_3$, (14) gives $\text{ADV}(\text{P}_3^2) = 0$, (15) gives $\text{ADV}(\text{P}_4^1) = 0$, and (18) confirms that $\text{ADV}(\text{P}_4) = 0$.
   Intermediate level: (15) gives $\text{ADV}(\text{P}_5^1) = 1079$, (19) gives $\text{ADV}(\text{P}_5) = 699.19$, (14) gives $\text{ADV}(\text{P}_5^2) = 699.19$, (15) gives $\text{ADV}(\text{P}_6^1) = 119$, (19) gives $\text{ADV}(\text{P}_6) = 102.82$, and (14) gives $\text{ADV}(\text{P}_6^2) = 102.82$.
   Lowest level: (15) gives $\text{ADV}(\text{P}_7^1) = 239$, (19) gives $\text{ADV}(\text{P}_7) = 154.87$, and (14) gives $\text{ADV}(\text{P}_7^2) = 154.87$; then (15) gives $\text{ADV}(\text{P}_8^1) = 719$, (19) gives $\text{ADV}(\text{P}_8) = 621.21$, and (14) gives $\text{ADV}(\text{P}_8^2) = 621.21$.

Output. Compound process latencies: $\ell_5^2 = 24.1\,\mu s$, $\ell_6^2 = 288.6\,\mu s$ and $\ell_2^2 = 34.752\,\mu s$; periods: $\text{PER}(\text{P}_5) = 37.04\,\mu s$, $\text{PER}(\text{P}_6) = 333.33\,\mu s$, $\text{PER}(\text{P}_7) = 154.3\,ns$ and $\text{PER}(\text{P}_8) = 462.96\,ns$. Burstinesses are large, $\text{ADV}(\text{P}_7) = 154.87$, $\text{ADV}(\text{P}_8) = 621.21$, $\text{ADV}(\text{P}_5) = 699.19$ and $\text{ADV}(\text{P}_6) = 102.82$, but this only reports a high variability around the average period. The clock's effect on resource usage is more significant: the parallelism degree is

$$\mathsf{maxpll}(\text{P}_7) = 2 \quad \text{and} \quad \mathsf{maxpll}(\text{P}_8) = 3.$$

This demonstrates how HPN achieve a precise description of the burst rate within an average periodic scheme. Finally, we get the bandwidth and buffer results:

$$\begin{array}{ll} \mathsf{maxbw}(\mathrm{P}_7^2\mathrm{P}_8^1) = 30\,Mpixel/s & \mathsf{maxbw}(\mathrm{P}_8^2\mathrm{P}_3^1) = 10\,Mpixel/s \\ \mathsf{maxbw}(\mathrm{P}_7^1) = 80\,Mpixel/s & \mathsf{maxbw}(\mathrm{P}_7^2) = 30\,Mpixel/s \\ \mathsf{maxbw}(\mathrm{P}_8^1) = 22.5\,Mpixel/s & \mathsf{maxbw}(\mathrm{P}_8^2) = 10\,Mpixel/s \end{array}$$

Eventually, we compute the buffer size for the atomic flat channel between the two filters again: message liveness is

$$\begin{aligned} d &= \max_{0 \le n < 1080} \left(\mathrm{ACT}\left(\mathrm{P}_6, \left\lceil (n+1)\frac{Q_5^2}{Q_6^1} \right\rceil - 1\right) + \ell_6 - \mathrm{ACT}(\mathrm{P}_5, n) \right) \\ &= \mathrm{ACT}(\mathrm{P}_6, 0) + \ell_6 - \mathrm{ACT}(\mathrm{P}_5, 0), \\ &= 192.2\,\mu s + 288.6\,\mu s - 0 \;=\; 480.8\,\mu s \end{aligned}$$

hence

$$\begin{aligned} \mathsf{maxbuf}(\mathrm{P}_5^2\mathrm{P}_6^1) &= Q_5^2 \mathsf{overlap}(\mathrm{P}_5, d)\mathsf{overlap}(\mathrm{P}_2, d) \\ &= 720 \times \lceil 0.4808 \times 41.667 \rceil \times 1 \;=\; 720 \times 21 \\ &= 15.12\,kpixel \end{aligned}$$

This buffer is 116.7% larger than a single stripe buffer between the two filters. This overhead was expected because the evaluation of the maximal buffer assumed worst-case conditions on message input/output delays (we assume that reading/writing the buffer takes up the whole activation time).

7 Extensions

Applicability of the previous model is vastly improved when considering three simple extensions. First of all, we provide two special kinds of atomic processes for multiplexing and demultiplexing streams. For example, such processes refine the modeling of a picture-in-picture application, where a rectangle of a larger frame is replaced by a downscaled frame from another video stream. Activation of a *splitter* process sends one message alternatively through each of its output ports, whereas activation of a *selector* process receives one message alternatively from each of its input ports. We proved that a periodic alternation of process ports preserves the periodic nature of message and activations events.

Moreover, it is quite natural to relax the periodic constraint on input streams, and only require an average periodicity. This is easily achieved in adding a burstiness parameter to source processes. We proved that conservative reference dates can be deduced from the "latest" execution scheme associated with the "latest" valid schedule of source processes (a periodic one).

Eventually, we consider cyclic networks whose semantics differs from iteration modeling in Petri nets: in stream-processing frameworks, cycles model feedback or data reuse. The latency for a message to traverse a given circuit must be less than or equal to the average period of the initiating process. In other words, a cyclic path is legal as long as it has *no* influence on the global throughput; it can be statically checked through a path constraint (analogue to the scaling rule) and an additional *bootstrap* constraint. Dynamic noise reduction is a typical example: a noise threshold is updated to provide dynamic control over the filtering stage.

8 Conclusion and Future Work

Multi-periodic process networks are an expressive model and a powerful tool for statically manipulating real-time properties, concurrency, and resource requirements of regular stream-processing applications. Primarily influenced by synchronous extensions to Kahn process networks, they exploit the application's regularity and hierarchical structure to extend the range of possible analyses and transformations. Six major properties can be expressed:

abstraction: nested processes allow for different levels of specification, both for messages (hierarchical nature of data structures) and activation events;

composition: the same property set describes all processes (latency, period...); nested processes are analyzed only once and reused through MPPN libraries;

synchronization: nesting of processes and hierarchical data-flow activation provides an elegant and efficient tool for modeling synchronization;

jitter: event dates — whether messages or process activations — are bounded within "earliest" and "latest" deterministic functions;

bursts: deterministic bursts of events can be captured explicitly within periodic event schemes, using hierarchical layers of periodic characterizations;

sequencing: communication uses First-In First-Out (FIFO) channels, but implicit reordering is allowed when assembling/splitting messages.

As a result, tradeoffs between synchronization (thus, concurrency), buffering and bandwidth requirements can be expressed and manipulated in a unified framework. This greatly simplifies the design and refinement of applications, providing quantitative feedback on resource usage early in the design process.

A prototype of the verifier was implemented in Java; it uses XML representations of MPPN for easy integration into application design environments. In the context of a software/hardware co-design project, we are carrying large-scale experiments to evaluate the benefits of MPPN for design space exploration and resource allocation. We envision a direct inference of MPPN parameters from a high-level language specification and a machine description.

Acknowledgments: We received many contributions from the further members of the SANDRA team: Christine Eisenbeis, Laurent Pasquier, Valerie Rivierre-Vier, Francois Thomasset and Qin Zhao. We thank them for their support and in-depth reviews.

References

1. F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat. *Synchronization and Linearity.* Wiley, 1992.
2. G. Berry, P. Couronné, and G. Gonthier. The synchronous approach to reactive and real-time systems. *Proc. IEEE*, 79(9):1270–1282, Sept. 1991.
3. B. Berthomieu and M. Diaz. Modelling and Verification of Time-Dependent Systems Using Time Petri Nets. *IEEE Trans. on Software Eng.*, 17, Mar. 1991.
4. J.-Y. L. Boudec and P. Thiran. *Network Calculus.* Springer LNCS 2050, Jan. 2002.
5. J. T. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. Ptolemy: A framework for simulating and prototyping heterogeneous systems. *J. Comp. Simulation*, 4, 1992.
6. P. Caspi and M. Pouzet. Synchronous Kahn Networks. In *ACM SIGPLAN Int. Conference on Functional Programming (ICFP)*, Philadelphia, May 1996. ACM.
7. G. Cohen, S. Gaubert, and J.-P. Quadrat. Algebraic system analysis of timed petri nets. *Idempotency, Cambridge University Press*, 1997.
8. G. Kahn. The Semantics of a Simple Language for Parallel Programming. In *IFIP 74 Congress*, Amsterdam, 1974. North-Holland.
9. B. Kienhuis, E. Rijpkema, and E. Deprettere. Compaan: Deriving process networks from matlab for embedded signal processing architectures. In *Proc. 8th workshop CODES*, pages 13–17, NY, May 3–5 2000. ACM.
10. E. A. Lee and J. C. Bier. Architectures for statically scheduled dataflow. *J. Parallel and Distributed Computing*, 10(4):333–348, Dec. 1990.
11. H. Leverge, C. Mauras, and P. Quinton. The ALPHA language and its use for the design of systolic arrays. *J. of VLSI Signal Processing*, 3:173–182, 1991.
12. P. Sénac and M. Diaz. Time Streams Petri Nets, A Model for Timed Multimedia Informations. In *16th Int. Conf. Appl. and Theory of Petri Nets*, Turin, June 1995.

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399