Multi-Periodic Process Networks: Prototyping and Verifying Stream-Processing Systems

Albert Cohen,[1] Daniela Genius,[1 2] Abdesselem Kortebi,[2] Zbigniew Chamski,[2] Marc Duranton[2] and Paul Feautrier[1]

[1] INRIA Rocquencourt, A3 Project [2] Philips Research

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.

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.

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.

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, 13, 8]. 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 [11, 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 [9] 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 [12] 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 [10]; it is also a powerful simulation tool. KPN model-

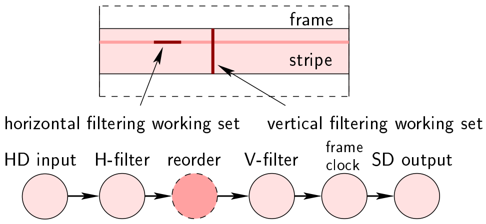

Fig. 1. Example: downscaler

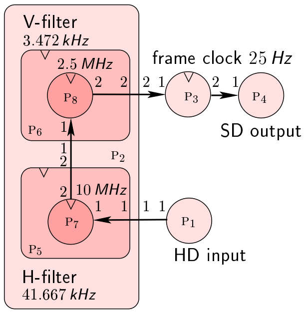

Fig. 2. Simple model of the downscaler

ing, 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 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 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 ($\textsc{act}, \textsc{msg}$), such that $\textsc{act} : \mathbf{P} \times \mathbb{N} \to \mathbb{R}$ maps processes and activation counts to activation dates, $\textsc{msg} : \{\textsc{p}_i^j \mid \textsc{p}_i \in \mathbf{P} \wedge j \in \textsc{in}(\textsc{p}_i) \cup \textsc{out}(\textsc{p}_i)\} \times \mathbb{N} \to \mathbb{R}$ maps process ports and message counts to message dates; $\textsc{act}(\textsc{p}_i, n)$ is the date of activation n of process $\textsc{p}_i$, $\textsc{msg}(\textsc{p}_i^j, n)$ is the date of message n hitting port $\textsc{p}_i^j$.

4.1 Propagation in atomic channels

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 $\textsc{p}_k$ through $\textsc{p}_i^j\textsc{p}_k^l$ correspond to $\lceil nQ_k^l/Q_i^j \rceil$ messages sent by $\textsc{p}_i$.

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

$$\forall \textsc{p}_i^j\textsc{p}_k^l \in \mathbf{C}_{\textsc{flat}} : \textsc{msg}(\textsc{p}_k^l, n) = \textsc{msg}\big(\textsc{p}_i^j, \lceil (n+1)Q_k^l/Q_i^j \rceil - 1\big) + c_{i,k}^{j,l}. \qquad (1)$$

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 $\textsc{p}_k^l$:

$$\forall \textsc{p}_i^j\textsc{p}_k^l \in \mathbf{C}_{\textsc{up}} : \textsc{msg}(\textsc{p}_k^l, n) = \textsc{act}(\textsc{p}_k, n) + \textsc{msg}\big(\textsc{p}_i^j, \lceil Q_k^l/Q_i^j \rceil - 1\big). \qquad (2)$$

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 $\textsc{p}_i$, this message is *not* propagated further on the channel (and possibly decomposed) before $\textsc{p}_i$ activates on this very message. Activation of $\textsc{p}_i$ coincides with the reception of a message at port $\textsc{p}_k^l$, since $Q_k^l \leq Q_i^j$ (decomposition into smaller messages). Therefore,

$$\forall \textsc{p}_i^j\textsc{p}_k^l \in \mathbf{C}_{\textsc{down}} : \textsc{msg}(\textsc{p}_k^l, n) = 0. \qquad (3)$$

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 $\textsc{p}_i$. A process $\textsc{p}_i$ enters activation n as soon as the following data-flow condition is met: every input port $j \in \text{IN}(\text{P}_i)$ 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:

$$\text{ACT}(\text{P}_i, n) = \max_{j \in \text{IN}(\text{P}_i)} \text{MSG}(\text{P}_i^j, n). \tag{4}$$

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:

$$\text{MSG}(\text{P}_i^j, n) = \text{ACT}(\text{P}_i, n) + \ell_i^j. \tag{5}$$

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

In the following, details and proofs that had to be left out can be found in [7]. 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 $j \in \text{IN}(\text{P}_i)$ and an output port $l \in \text{OUT}(\text{P}_i)$. If these ports are connected through a path π of channels and sub-processes of P_i, one single message hitting P_i^j may traverse several assembling/splitting stages through π, but it must yield one single message hitting P_i^l. This is intrinsic to the hierarchical model; the following path constraints enforce the scaling rule:[1]

$$\prod\nolimits_{\text{P}_i^j \text{P}_k^l \in \pi} (Q_i^j / Q_k^l) = 1 \text{ and } \forall \pi' \text{ prefix of } \pi : \prod\nolimits_{\text{P}_i^j \text{P}_k^l \in \pi'} (Q_i^j / Q_k^l) \geq 1. \tag{6}$$

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 $\text{ACT}(\text{P}_i, 0)$ — the *reference date* — and $\text{PER}(\text{P}_i)$ — the *period*,

$$\text{ACT}(\text{P}_i, n) = \text{ACT}(\text{P}_i, 0) + n\text{PER}(\text{P}_i). \tag{7}$$

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 (6) which ensures that any single activation of P_i sends exactly one message through P_i^j. We may thus extend (5) to *compound* processes: $\text{MSG}(\text{P}_i^j, n) - \text{ACT}(\text{P}_i, n)$ is a constant, ℓ_i^j can be computed for $n = 0$ based on information at the lower level:

$$\forall \text{P}_i \in \mathbf{P}, j \in \text{OUT}(\text{P}_i) : \ell_i^j = \text{MSG}(\text{P}_i^j, 0) - \text{ACT}(\text{P}_i, 0). \tag{8}$$

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

[1] This equation applies when messages are multiples of one another.

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. A clocked process P_i is characterized through a *clock frequency* f_i such that $f_i \geq 1/\mathrm{PER}(\mathrm{P}_i)$; equality enforces periodicity (e.g., to enable stream resynchronization for video output).

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 have proven that all processes follow a steady-state scheme which extends and relaxes the periodic constraint (7) on source processes; starting from (7), this follows inductively from (1), (4) and (5). For each process P_i, there exist an *average period* $\mathrm{PER}(\mathrm{P}_i)$ and a *burstiness* $\mathrm{ADV}(\mathrm{P}_i)$ such that

$$\forall n \geq 0 : (n - \mathrm{ADV}(\mathrm{P}_i))\mathrm{PER}(\mathrm{P}_i) \leq \mathrm{ACT}(\mathrm{P}_i, n) - \mathrm{ACT}(\mathrm{P}_i, 0) \leq n\mathrm{PER}(\mathrm{P}_i) \quad (9)$$

$$\forall n \geq 0 : (n - \mathrm{ADV}(\mathrm{P}_i^j))\mathrm{PER}(\mathrm{P}_i) \leq \mathrm{MSG}(\mathrm{P}_i^j, n) - \mathrm{MSG}(\mathrm{P}_i^j, 0) \leq n\mathrm{PER}(\mathrm{P}_i). \quad (10)$$

The burstiness is the maximal number of *advance* activations of P_i, i.e., 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 $\mathrm{ACT}(\mathrm{P}_i, n)$ can be hoped for (possibly exact ones): deterministic event bursts can be characterized effectively within the relaxed periodic scheme.

Considering an output port j of a process P_i, messages hit j after a constant delay, hence output message burstiness is equal to activation burstiness:

$$\forall j \in \mathrm{OUT}(\mathrm{P}_i) : \mathrm{ADV}(\mathrm{P}_i^j) = \mathrm{ADV}(\mathrm{P}_i). \quad (11)$$

This equation stands for both *upward* channels (in compound processes) and atomic processes.

Sending one message and $\text{ADV}(\mathrm{P}_i^j)$ advance messages at port P_i^j corresponds to sending $Q_i^j(1+\text{ADV}(\mathrm{P}_i^j))$ bytes of data. These data are received as one message plus $\text{ADV}(\mathrm{P}_k^l)$ advance messages at port P_k^l, i.e., $Q_k^l(1+\text{ADV}(\mathrm{P}_k^l))$ bytes. For *flat* channels, the result is the following:

$$\forall \mathrm{P}_i^j\mathrm{P}_k^l \in \mathbf{C}_{\text{FLAT}} : Q_i^j(1+\text{ADV}(\mathrm{P}_i^j)) = Q_k^l(1+\text{ADV}(\mathrm{P}_k^l)); \tag{12}$$

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

$$\forall \mathrm{P}_i^j\mathrm{P}_k^l \in \mathbf{C}_{\text{DOWN}} : Q_i^j = Q_k^l(1+\text{ADV}(\mathrm{P}_k^l)). \tag{13}$$

Notice that (12) and (13) 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:

$$\forall \mathrm{P}_i^j\mathrm{P}_k^l \in \mathbf{C} : Q_i^j/\text{PER}(\mathrm{P}_i) = Q_k^l/\text{PER}(\mathrm{P}_k). \tag{14}$$

Activation burstiness can be deduced from the data-flow scheme, replacing $\text{ACT}(\mathrm{P}_i, n)$ and $\text{MSG}(\mathrm{P}_i^j, n)$ by their lower bounds in (4). The result is that processes tend to "smooth" message bursts and initiation delays:

$$\text{ADV}(\mathrm{P}_i) = \min_{j\in\text{IN}(\mathrm{P}_i)} \left\{ \text{ADV}(\mathrm{P}_i^j) + \frac{\text{ACT}(\mathrm{P}_i, 0) - \text{MSG}(\mathrm{P}_i^j, 0)}{\text{PER}(\mathrm{P}_i)} \right\}. \tag{15}$$

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. If the process is clocked, the former result is multiplied by $1 - 1/(f_i\text{PER}(\mathrm{P}_i))$; one expectedly get $\text{ADV}(\mathrm{P}_i) = 0$ when $\text{PER}(\mathrm{P}_i) = 1/f_i$.

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: $\ell_i = \max_{j\in\text{OUT}(\mathrm{P}_i)} \ell_i^j$. 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):

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

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:

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

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, see [7] for details. 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, whereas channel bandwidth provides some insight about network contention when focusing on distributed architectures.

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.

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.

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 $\mathrm{ACT}(\mathrm{P}_i, 0)$ for each source process (e.g., 0), period $\mathrm{PER}(\mathrm{P}_i)$ for one process per weakly-connected component of the network, burstiness $\mathrm{ADV}(\mathrm{P}_i)$ 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 (6).
2. Compute $\mathrm{PER}(\mathrm{P}_i)$ traversing the network incrementally, starting from processes with known periods using (14) 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), (4) and (5);
   - deduce latency ℓ_i^j for every output port j, using (8).
4. Compute $\mathrm{ADV}(\mathrm{P}_i)$ and $\mathrm{ADV}(\mathrm{P}_i^j)$ through a top-down traversal, following the topological ordering at each hierarchical level, using (11), (12) and (15).

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: $\mathrm{ACT}(\mathrm{P}_1, 0) = 0$, $\mathrm{ADV}(\mathrm{P}_1) = 0$, $\mathrm{PER}(\mathrm{P}_4) = 40\,ms$ $(25\,Hz)$,

$\text{ADV}(\text{P}_4) = 0$, $\ell_1^1 = 0$ (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: $f_5 = f_7 \times (3/720) = 41.67\,kHz$, $f_6 = f_8 \times (4/(4 \times 720)) = 3.47\,kHz$.

Output. Compound process latencies: $\ell_5^2 = 24.1\,\mu s$, $\ell_6^2 = 288.6\,\mu s$ and $\ell_2^2 = 34.752\,ms$; periods: $\text{PER}(\text{P}_5) = 37.04\,\mu s$, $\text{PER}(\text{P}_6) = 333.33\,\mu s$, $\text{PER}(\text{P}_7) = 154.3\,ns$, $\text{PER}(\text{P}_8) = 462.96\,ns$. Burstinesses are large, $\text{ADV}(\text{P}_7) = 154.9$, $\text{ADV}(\text{P}_8) = 621.2$, $\text{ADV}(\text{P}_5) = 699.2$ and $\text{ADV}(\text{P}_6) = 102.8$, 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 $\mathsf{maxpll}(\text{P}_7) = 2$ and $\mathsf{maxpll}(\text{P}_8) = 3$. This demonstrates how MPPN achieve a precise description of the burst rate within an average periodic scheme. Finally, we get the bandwidth and buffer results: $\mathsf{maxbw}(\text{P}_7^1) = 80\,Mpixel/s$, $\mathsf{maxbw}(\text{P}_7^2) = 30\,Mpixel/s$, $\mathsf{maxbw}(\text{P}_8^1) = 22.5\,Mpixel/s$, $\mathsf{maxbw}(\text{P}_8^2) = 10\,Mpixel/s$ and $\mathsf{maxbuf}(\text{P}_5^2\text{P}_6^1) = 15.12\,kpixel$, which expectedly corresponds to a single stripe buffer between the two filters plus 116.7% overhead, making conservative assumptions on message liveness.

7 Extensions, Conclusion and Future Work

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.

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.

A prototype of the verifier was implemented in Java; it uses XML representations of MPPN for easy integration into application design environments.

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. A. Cohen and D. Genius. Multi-periodic process networks: Technical report. http://www-rocq.inria.fr/~acohen/publications/mppn, 2002.
8. G. Cohen, S. Gaubert, and J.-P. Quadrat. Algebraic system analysis of timed petri nets. *Idempotency, Cambridge University Press*, 1997.
9. G. Kahn. The Semantics of a Simple Language for Parallel Programming. In *IFIP 74 Congress*, Amsterdam, 1974. North-Holland.
10. 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.
11. E. A. Lee and J. C. Bier. Architectures for statically scheduled dataflow. *J. Parallel and Distributed Computing*, 10(4):333–348, Dec. 1990.
12. 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.
13. 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.