Transaction Parameterized Dataflow: A Model for Context-Dependent Streaming Applications

Xuan Do, Stephane Louise, Albert Cohen

Transaction Parameterized Dataflow: A Model for Context-Dependent Streaming Applications

Xuan Khanh Do, Stephane Louise
CEA, LIST
91191 Gif-sur-Yvette Cedex, France
Email: xuankhanh.do@cea.fr, stephane.louise@cea.fr

Albert Cohen
INRIA and ENS
45 rue d'Ulm, 75005 Paris, France
Email: albert.cohen@inria.fr

***Abstract*—Static dataflow programming models are well suited to the development of embedded many-core systems. However, complex signal and media processing applications often display dynamic behavior that do not fit the classical static restrictions. We propose Transaction Parameterized Dataflow (TPDF), a new model of computation combining integer parameters—to express dynamic rates—and a new type of control actor—to allow topology changes and time constraints enforcement. We present static analyses for liveness and bounded memory usage. We also introduce a static scheduling heuristic to map TPDF to massively parallel embedded platforms. We validate the model and associated methods using a cognitive radio application, demonstrating significant buffer size and performance improvements compared to state of the art models including Cyclo-Static Dataflow (CSDF).**

I. Introduction

The broader availability of low-power many-core platforms—such as the MPPA-256 chip from Kalray (256 cores) [1] or Epiphany from Adapteva (64 cores)—opens new opportunities for system designers. The complexity of these platforms also pushes for practical solutions accessible to domain experts, facilitating efficient mapping, performance tuning and analysis of applications.

Programming languages based on dataflow models of computation have emerged as a comprehensive solution towards the automation of embedded system design. Dataflow application are modeled as a directed graph where nodes represents actors (iterated execution of tasks) and edges represent communication channels. Among these, decidable dataflow models in the SDF [2] or CSDF [3] family are useful for their predictability, formal abstraction, and amenability to powerful optimization techniques. However, for signal processing applications, it is not always possible to represent all of the functionality in terms of purely decidable dataflow representations; typical challenges include variable data rate processing, multi-standard or multi-mode signal processing operation, and data-dependent forms of adaptive signal processing behavior. For this reason, numerous dynamic dataflow modeling techniques—whose behavior is characterized by dynamic variations in resource requirements—have been proposed. In many of these, in exchange for the increased modeling flexibility (high expressive power) provided by the underlying techniques, one must give up guarantees on compile-time buffer underflow (deadlock) and overflow validation (boundedness).

In this paper, we introduce a new dynamic Model of Computation (MoC), called *Transaction Parameterized Dataflow* (TPDF), allowing variable production and consumption rates and dynamic changes of the graph topology. TPDF is designed to be statically analyzable regarding the essential deadlock and boundedness properties, while avoiding the aforementioned restrictions of decidable dataflow models.

The remainder of this paper is organised as follows. Section II-A recalls some background on Cyclo-Static Dataflow (CSDF) needed to introduce our parametric and dynamic extension in Section II-B. Section III presents the static analyses for liveness and boundedness and a scheduling heuristic for this model, which is illustrated in Section IV-B by a case study. Finally, we give an overview of the related models in Section V and summarize our contributions in Section VI.

II. Model of Computation

Let us first recall the principles of CSDF [3], one of the reference dataflow MoC for applications in the signal processing domain. Then, we introduce our model as a parameterized extension of CSDF with transaction processes.

A. Basic Model: CSDF

In this work, we choose Cyclo-Static Dataflow (CSDF) [3] as the base model for TPDF because it is deterministic and allows for checking conditions such as deadlocks and bounded memory execution at compile/design time, which is usually not possible for Dynamic Dataflow (DDF). In CSDF, a program is defined as a directed graph $G = \langle A, E \rangle$, where A is a set of actors, $E \subseteq A \times A$ is a set of directed communication channels. Actors represent functions that transform the input data streams into output data streams. An atomic piece of data carried by a channel is called a $token$. Each channel has an initial status, characterized by its *initial tokens*.

Each actor $a_j \in A$ has a cyclic *execution sequence* of length τ_j, $[f_j(0), \cdots, f_j(\tau_j - 1)]$ which can be understood as follows: The n-th time that actor a_j is fired, it executes the code of function $f_j(n \mod \tau_j)$ and produces (consumes) $x_j^u(n \mod \tau_j)$ (or $y_j^u(n \mod \tau_j)$) tokens on its output (input) channel e_u. The firing rule of a cyclo-static actor a_j is evaluated as true for its n-th firing if and only if all input channels contain at least $y_j^u(n \mod \tau_j)$ tokens. The total number of tokens produced (consumed) by actor a_j on channel e_u during the first n invocations, denoted by $X_j^u(n) = \sum_{l=0}^{n-1} x_j^u(l)$ (or $Y_j^u(n) = \sum_{l=0}^{n-1} y_j^u(l)$).

One of the most important properties of the CSDF model is the ability to derive at compile-time a schedule for the actors.

Definition 1. *Given a connected CSDF graph G, a* ***valid static schedule*** *for G is a schedule that can be repeated infinitely while the buffer size remains bounded. A vector*

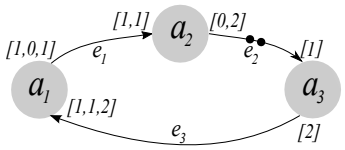


Fig. 1. Figure 1 shows a CSDF graph consisting of four actors and four communication channels. Edge e_2 has two initial tokens.

*$\overrightarrow{q} = [q_1, q_2, ..., q_n]^T$ is a **repetition vector** of G if each q_j represents the number of invocations of an actor a_j in a valid static schedule for G. A CSDF graph is called **consistent** if and only if it has a non-trivial repetition vector [3].*

Theorem 1. *In a CSDF graph, a repetition vector $\overrightarrow{q} = [q_1, q_2, ..., q_n]^T$ is given by [3]:*

$$\overrightarrow{q} = P \cdot \overrightarrow{r} \quad , with \quad P = P_{jk} = \begin{cases} \tau_j & , if \ \ j = k \\ 0 & , otherwise \end{cases} \tag{1}$$

where $\overrightarrow{r} = [r_1, r_2, ..., r_n]^T$ is a solution of

$$\Gamma \cdot \overrightarrow{r} = 0 \tag{2}$$

and where the topology matrix Γ is defined by

$$\Gamma_{uj} = \begin{cases} X_j^u(\tau_j) & , if \ task \ a_j \ produces \ on \ edge \ e_u \\ -Y_j^u(\tau_j) & , if \ task \ a_j \ consumes \ from \ edge \ e_u \\ 0 & , otherwise \end{cases} \tag{3}$$

In Figure 1, the solutions are $\overrightarrow{q} = [3, 2, 2]^T$. It can only start by firing a_3 twice; then, a_1 fired three times, and finally a_2 twice. Since each actor has been fired the exact number of times required by its solution, a schedule can be found. We represent it as the string $(a_3)^2(a_1)^3(a_2)^2$ where the superscripts denote repetition count.

B. Transaction Parameterized Dataflow Model

We extend CSDF by allowing rates to be *parametric* and a new type of *control* actor, channel and port. For a compact formal notations, we assume in that paper that kernels, which play the same role as computation units (actors) as in CSDF, have at most *one* control port. Kernels without control ports are considered to always operate in a *dataflow* way, i.e., a kernel starts its firings only when there is enough data tokens on all of its data input ports.

Definition 2. *A TPDF graph $\mathcal{G}$ is defined by a tuple (K, G, E, P, R_k, R_g, α, ϕ^*) where:*

- *K is a non-empty finite set of kernels and G is a finite set of control actors such that $K \cap G = \varnothing$. For each kernel $k \in K$, M_k denotes the set of modes indicated by the control node connected to its unique control port c. The following modes are available within a TPDF graph:*
  - *Select one of the data inputs (outputs)*
  - *Select more than one data input (output)*
  - *Select available data input with the highest priority*
  - *Wait until all data inputs are available*

  In this context, the effect of control tokens can be also described as selecting data input and output ports besides choosing modes. Indeed, at a given time, the input and output ports of an edge may be in a different state. However, it does not affect the firings of kernels or control actors, only the data tokens that are chosen or rejected.
- *$E \in O \times (I \cup C)$ is a set of directed channels, where I, C, O is the union of all input, control and output port sets respectively. $E_c = E \backslash (O \times I)$ is the set of all control channels. A control channel can start only from a control actor and is connected to a control port.*
- *P is a set of integer parameters.*
- *$R_k : M_k \times (I_k \cup C_k \cup O_k) \times \mathbb{N} \longrightarrow \mathbb{N}$ assigns the rate to the ports of the n-th firing of k for each mode. The rate $R_k(m, c, n) = \{0, 1\}$ for all modes $m \in M_k$ and for all firings of k.*
- *$R_g : (I_g \cup C_g \cup O_g) \times \mathbb{N} \longrightarrow \mathbb{N}$ assigns the rate to each port of the n-th firing of a control actor g.*
- *$\alpha : (I \cup C \cup O) \longrightarrow \mathbb{N}$ returns for each port its priority.*
- *$\phi^* : E \longrightarrow \mathbb{N}$ is the initial channel status.*

In this paper, we assume that a kernel $k \in K$ waits until its control port becomes available to be fired by reading one token from this port. This token defines in which mode $m \in M_k$ in which k will operate. One of the most important property of TPDF, which is different from SDF and CSDF, it is that a kernel or a control actor does not have to wait until sufficient tokens are available at every data input port. This new property allows the capacity of reconfiguration of graphs depending on context and time.

The n-th firing of a control actor $g \in G$ starts by waiting until $R_g(i, n)$ and $R_g(c, n)$ tokens are available at every input $i \in I_g$ and $c \in C_g$, where $R_g(c, n) = \{0, 1\}$. After performing its actions, the n-th firing of g ends by removing the $R_g(i, n)$ and $R_g(c, n)$ tokens from its input data and control ports and writing $R_g(o, n)$ tokens to each output control port $o \in O_g$.

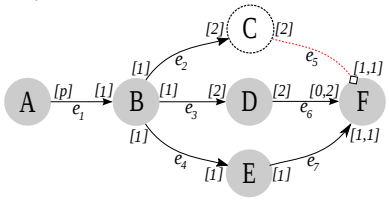


Fig. 2. A simple TPDF graph with integer parameter p and control actor C

Example 1. *Figure 2 shows a simple TPDF graph where actors have constant or parametric rates (e.g., p for the output rate of A). The repetition vector is $[2, 2p, p, p, 2p, 2p]$. C is a control actor and e_5 is a control channel. A sample execution of the graph is the following: A fires and produces p tokens on e_1. Then B fires and produces one token on edge e_2, e_3, e_4. Only E can fire because there are enough tokens on its input edge and produce one token on edge e_7. B (and A if necessary) will fire a second time and produce another token on edge e_2, e_3, e_4. Then C, D and E will fire and produce 2, 2 and 1 token, respectively, on edge e_5, e_6, e_7. Finally, F fires two times, each time it consumes one token from its control port. This token determines in which mode F will be fired. In this case, F can choose two tokens from e_6 or one from e_7 and remove remaining tokens. This continues until each actor has fired a number of times equal to its repetition count.*

In TPDF, we define 2 data distribution kernels *Select-duplicate*, *Transaction Box* [4] and a new type of control clock in a dataflow way.

a) Select-duplicate: kernels with *one* entry and n outputs (n is a maximum number for automatic sizing). At a given time any combination of the n outputs can be enabled. Each time a data token is read on the input line, it is copied on this combination of the n output channels.

b) Transaction: symmetric processes with n inputs and *one* output. Its role is to atomically select a predefined number of tokens from one or several of its input to its output. By using special modes predefined by TPDF and combining with a control actor, the Transaction process implements important actions not available in usual dataflow MoC: Speculation, Redundancy with vote, Highest priority at a given deadline, Selection of an active data-path among several [4].

c) Clock: can be considered as a watchdog timer with control tokens sent each time there is a timing out.

III. Static Analyses

This section presents the three static analyses needed to ensure consistency, boundedness and liveness of TPDF graphs. In Section III-A, we check rate consistency by adapting the analysis of CSDF to TPDF. Conditions for rate consistency and solutions of balanced equations are computed in terms of symbolic expressions. In Section III-B, we check that, along with rate consistency, the TPDF MoC with control actor and parameter setting ensures boundedness. Section III-C completes the analysis chain by checking for liveness.

A. Rate consistency

As in SDF and CSDF, we check the rate consistency of a TPDF graph by generating the associated system of balance equations expressed in matrix-form as in Equation (2) and this system must be shown to have a non null solution for all possible values of parameters, all possible reconfigurations of the graph and all modes of the kernels.

The matrix is generated by considering the parametric rates and by ignoring all possible configurations of the graph. If the system is rate consistent when all edges are present, then it is also consistent when one or more several edges are removed. Indeed, when removing edges (i.e., tokens produced in this edge are not used and will be rejected), the resulting system of balance equations will be a subset of the system of equations of the fully connected graph. Checking rate consistency of all edges maybe considered too strict because it does not take into account the fact that some input edges may not be active in the same mode. However, it simplifies the understanding and implementation since the graph has a unique iteration vector.

The minimal solutions for all actors when solving the system of equations (2) are found by eliminating all the coefficients or parametric factors common to all solutions. Then, we arbitrarily set one of the solutions to 1 and recursively find other solutions. Finally, we normalize the solutions to integers. If the system of equations (2) has a non-trivial solution, the TPDF graph also satisfies the necessary and sufficient conditions for the existence of parametric solutions, introduced for parametric models such as SPDF [5] and BPDF [6].

Example 2. *For the graph of Figure 2, by setting $r_A = 1$, we consecutively get:*

$$r_B = p, r_C = p/2, r_D = p/2, r_E = p, r_F = p/2 \quad (4)$$

To normalize the fractional solutions we multiply by 2. Finally, we get the vector $\vec{r}$ and also the repetition vector $\vec{q}$ by multiplying with the matrix P:

$$\vec{r} = [2, 2p, p, p, 2p, p]\,, \vec{q} = [2, 2p, p, p, 2p, 2p] \quad (5)$$

and a possible valid static schedule for this graph is $A^2B^{2p}C^pD^pE^{2p}F^{2p} = A^2(B^2CDE^2F^2)^p$.

B. Boundedness

Without dynamic topology changes, rate consistency is sufficient to ensure that a graph returns to its initial state after each iteration and boundedness is guaranteed. However, in TPDF, a graph can change its topology within a valid static schedule by using the control actor. Yet, not all static schedules are safe and their boundedness must be checked. The criterion ensuring that reconfiguration by using control actor and several modes for one kernel are safe relies on the notion of *control area*. Intuitively, the criterion states that each control actor will be fired once per local iteration of the area it reconfigures.

Definition 3. *(Control area): The area of a control actor $g \in G$, noted Area(g), is defined as:*

$$Area(g) = \boldsymbol{prec}(g) \cup \boldsymbol{succ}(g) \cup \boldsymbol{infl}(g) \quad (6)$$

where

$$\begin{aligned} \boldsymbol{succ}(g) &= \{a_i \in (K \cup G) : \exists e_u = (g, a_i) \in E\} \\ \boldsymbol{prec}(g) &= \{a_i \in (K \cup G) : \exists e_u = (a_i, g) \in E\} \end{aligned} \quad (7)$$

and $\boldsymbol{infl}(g) = (\boldsymbol{succ}(\boldsymbol{prec}(g)) \cap \boldsymbol{prec}(\boldsymbol{succ}(g)))\backslash\{g\}$ is the list of actors between $\boldsymbol{prec}(g)$ and $\boldsymbol{succ}(g)$, influenced by g.

The *control area* of g is the set containing its sources, kernels or controls that receive its control tokens and all other influenced actors between these actors.

Definition 4. *(Local Solution) The local solution of an actor (kernel or control actor) a_i in a subset of actors $Z = \{a_1, \ldots, a_n\}$, written $q^L_{a_i}$, is defined as:*

$$q^L_{a_i} = \frac{q_{a_i}}{q_{\mathcal{G}}(Z)} \quad (8)$$

where $q_{\mathcal{G}}(Z) = gcd(q_{a_i}/\tau_i)\ \forall a_i \in Z$ (gcd denotes the greatest common divisor). Local solutions can be considered as a repetition vector for a subset of actors.

Definition 5. *(Rate Safety): A TPDF graph is rate safe if and only if, for each control actor $g \in G$ and each actor $a \in Area(g)$, the consumption and production rates of these actors ensure that, during a local iteration of its area, a control actor will be fired only one time. This condition is guaranteed, if and only if, for each control actor $g \in G$ and for each actor $a_i \in \boldsymbol{prec}(g) \cup \boldsymbol{succ}(g)$, connected with g by the channel e_u:*

$$\begin{cases} X^u_g(1) = Y^u_i(q^L_{a_i}) & \text{if } g \text{ is the production actor} \\ Y^u_g(1) = X^u_i(q^L_{a_i}) & \text{if } g \text{ is the consumption actor} \end{cases} \quad (9)$$

Rate safety ensures that, during a local iteration of an area of a given control actor, the total number of tokens consumed (produced) on any edge connected with the control actor is sufficient for this actor to fire exactly one time. It is ensured by a simple syntactic check on TPDF graphs.

Example 3. *In Figure 2, Area$(C) = \{B, D, E, F\}$ and possible static schedules for this graph are $A^2B^{2p}C^pD^p$*

$E^{2p}F^{2p}$ *and* $A^2(B^2CDE^2F^2)^p$*, where* $B^2CDE^2F^2$ *is a local solution for the Area*(C)*.* C *will be fired only one time for each iteration of this local area.*

Theorem 2. *(Boundedness) A rate consistent, safe and live TPDF graph returns to its initial state at the end of its iteration. Hence it can be scheduled in bounded memory.*

Proof. There are several modes defined by a control actor as defined in Section II-B. In fact, the Rate safety ensures that, during a local iteration of a control area, its influenced kernels receive synchronous control tokens, calculated by only one firing of the control actor and define in which mode this kernel will fire. For the modes that choose between the data inputs (e.g., Transaction), the dependence with kernels which produce these input tokens is not broken here because unchosen data input are considered only as not to be used. For the modes which choose between data outputs (e.g., Select-duplicate), we assume that there is a virtual control actor and a virtual kernel which chooses between data inputs, as in Figure 3. □

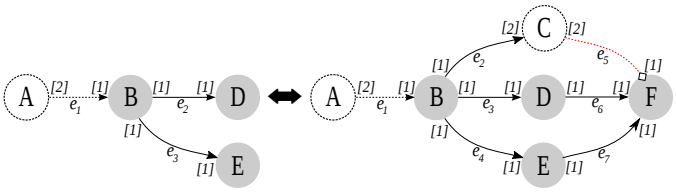


Fig. 3. B is a Select-duplicate to choose between D and E. This graph is equivalent with the second graph by adding two virtual actors: C and F, which consume data outputs from C, D and E. When B must choose between its data outputs, it sends a signal token to C, this control actor will send a control token to F, which choose only the data-paths chosen by B. In this case, boundedness of the graph is guaranteed.

Rate consistency and safety were crucial to ensure that the graph returns to its initial state after an iteration. However, we assumed that actors can be fired in the right order to respect dataflow constraints. This holds only when the graph is live and the next section shows how it is checked.

C. Liveness

In SDF and CSDF, checking liveness is performed by finding a schedule for a basic iteration. For TPDF, the situation is more complex for two reasons:

- First, control tokens change the topology of the graph by the selection or removal of data tokens within the iteration. However, this selection does not introduce new constraints among firings of control and kernel actors. Then, the topology change of the graph is not a reason which may introduce deadlocks.
- Second, actors may have to be fired a parametric number of times during an iteration. Finding a schedule may, in general, involve some inductive reasoning. We resolve this problem by using the following analysis.

A (C)SDF and TPDF graph deadlocks only if it contains at least one cycle. To deal with such problems, we use the standard clustering described in [7]. Given a connected, consistent TPDF graph $\mathcal{G}$, a subset $Z \subseteq (K \cup G)$, clustering Z involves replacing Z by a single actor Ω. The new actor

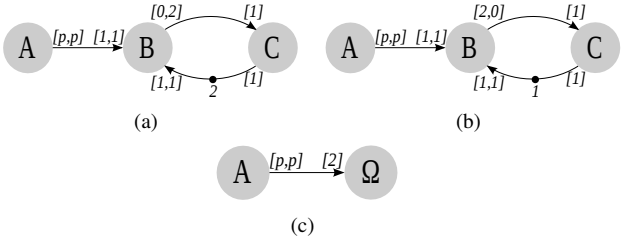


Fig. 4. (a), (b): Live TPDF graphs; (c): New graph obtained by clustering the cycle $Z = (B, C)$ into a single new actor Ω

Ω is connected to the same external ports as Z was, but the port rate and execution sequence must be adjusted. The rate sequence of each port of an actor $a_i \in Z$ connected to the rest of the graph is replaced by $r = Y_i^u(q_{a_i}^L)$ if e_u connected to an input port of a_i or $r = X_i^u(q_{a_i}^L)$ if e_u connected to an output port of a_i. It follows that a firing of Z corresponds to $q_{a_i}^L$ firings of its actors. Here, by clustering only cycles, we resolve the problem of liveness by checking the local solution of the clustered cycles. For the example in Figure 4(a), by clustering the cycle $Z = (B, C)$ into a new actor Ω to get the new graph in Figure 4(c). By resolving the balance equation (2), we find the schedule $A^2\Omega^p$ which corresponds to $A^2(B^2C^2)^p$ for the original graph (with $q_{\mathcal{G}}(Z) = p$, $q_B^L = 2$ and $q_C^L = 2$), which is therefore correctly found to be live.

A final refinement is needed to take into account the case in Figure 4(b). For instance, the previous refined algorithm would fail to find a schedule for the cycle with only one initial token. However, it is clearly live with the schedule $A^2(BC)^{2p}$ or $A^2(BCCB)^p$. In this case, we construct a *Late schedule* [8], which introduces the execution order between actors in CSDF and TPDF graphs. For instance, the cycle in Figure 4(b) has a local Late schedule $(BCCB)$ by applying the Algorithm presented in [8] for q_C^L firings of C. Then, the original graph is live with the global schedule A^2Z^p or $A^2(BCCB)^p$.

D. Scheduling

In Section III-C, we described a way to find a sequential schedule for TPDF applications. However, our objective is to use TPDF to implement streaming applications on many-core platforms with highly parallel schedules such as the MPPA-256 [1] clustered architecture, from Kalray, comprising 256 cores. The native dataflow programming model developed in ΣC (Sigma-C) for MPPA-256 uses the notion of *canonical period* [9], which represents the partial order corresponding to the execution of one iteration of the application in the form of a graph whose vertices are, for each task a_i, the q_i first occurrences of this task and channels are dependencies between these occurrences.

For TPDF graphs, we reuse this notion of canonical period with several changes in the scheduling techniques:

- The control actor is scheduled for execution with the highest priority (i.e., if there are several kernels and a control actor available concurrently, the control actor is ensured to have a processing unit available before the others). Message passing time must be accounted for within the scheduling so that the system acts as if it was instantaneous.

- The kernel which receives the control token is fired immediately after receiving the control token. If this kernel has to wait until its input data tokens are available, it passes into a sleeping queue and wakes up when there is enough tokens in its inputs ports, as requested by the control token. If this kernel is fired in a mode where several of its input ports are rejected, the scheduler uses the Actor Dependence Function [8] which defines the dependency between actors' executions to stop unnecessary firings.

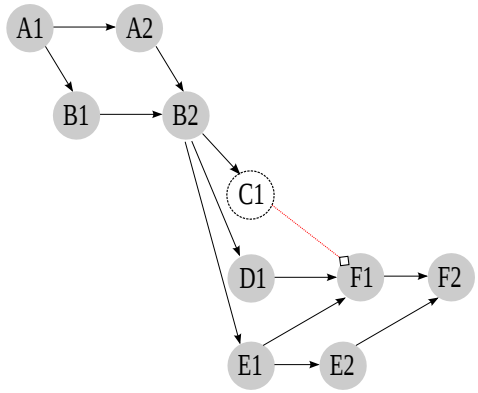


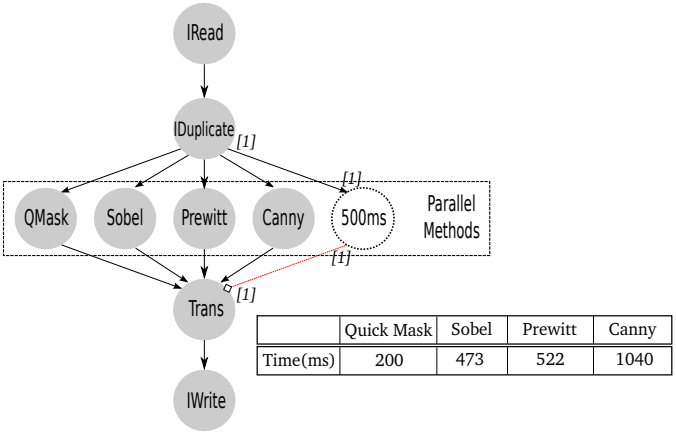
Fig. 5. *Canonical period* of the example graph in Figure 2 for $p = 1$. Number next to the name of each actor is its ordinal number of execution. *C1* is mapped onto a separate processing element. *F1* and *F2* are fired immediately after receiving the control tokens. Depending on this control token, it decides to wait until its input data available or not.

IV. Case Studies

A. Case-study on Edge Detection

Edge detection is one of the most significant tasks in image processing systems with various proposed algorithms: Quick Mask, Sobel, Prewitt, Kirsch, Canny. Gradient based edge detectors like Prewitt and Sobel are relatively simple and easy to implement, but are very sensitive to noise. The Canny algorithm is an optimal solution to problem of edge detection which gives better detection specially in presence of noise, but it is time consuming and require a lot of parameter setting. As can be seen in Figure 6, the execution time of Quick Mask is the shorter and Canny is the longest (tested on a standard Intel Core i3@2.53GHz for a 1024 $\times$ 1024 image).

In an ideal world, a programmer would use the best algorithm and be done with it, but possible real-time constraints can mitigate this idyllic view. When dealing with timing constraint, an average quality result at the right time is far better than an excellent result, later. The Canny filter may be the best algorithm for edge detections, but the execution time depends on the input image. In contrast, Quick Mask or Sobel have image-independent execution time (i.e., depending only on the size of the input image, not on its contents). So we can use a control actor of type clock to implement this time constraint, as can be seen in Figure 6. The *IRead* actor reads images from source and duplicates this image to be tested by different Edge Detection methods: Quick Mask, Sobel, Prewitt, Canny,... and its output is connected directly to the Transaction kernel. This box will select the best results in accordance with the deadline, implemented by using a control token received from the control actor every $500ms$. This kind of time-dependent decision is not available in usual CSDF. By contrast, our model fits well the case by using control actors.



	Quick Mask	Sobel	Prewitt	Canny
Time(ms)	200	473	522	1040

Fig. 6. TPDF graph of the Edge Detection application. Omitted rates equal to the image size $p \times q$. Execution time of different methods are measured for a 1024 $\times$ 1024 image. At the deadline, the best result will be chosen, according to the order: Canny $>$ Prewitt $>$ Sobel $>$ Quick Mask.

B. Case-Study on Cognitive Radio

We have applied the TPDF approach to an OFDM demodulator from the domain of cognitive radio, which is one of the fundamental subsystems of LTE and WiMAX wireless communication systems. Figure 7 illustrates a runtime-reconfigurable OFDM demodulator that is modeled as a TPDF graph. Here, actor *SRC* represents a data source that generates random values to simulate a sampler. In a wideband OFDM system, information is encoded on a large number of carrier frequencies, forming an OFDM symbol stream. In baseband processing, a symbol stream can be viewed in terms of consecutive vectors of length N. The symbol is usually padded with a cyclic prefix (CP) of length L to reduce inter-symbol inference (ISI) [10]. In Figure 7, the CP is removed by actor *RCP*. Then, actor *FFT* performs a fast Fourier transform (FFT) to convert the symbol stream to the frequency domain. This kernel is connected to a M-ary *QAM* demodulation, with a configurable *QPSK* configuration ($M = 2$ or $M = 4$). Finally, the output bits are collected by the data sink *SNK*.

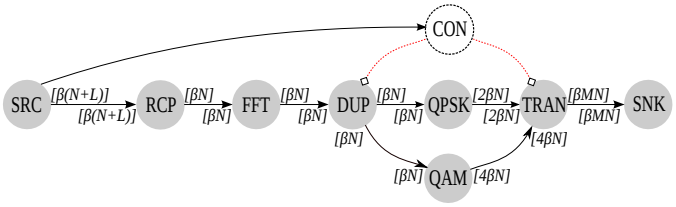


Fig. 7. TPDF model of a OFDM demodulator with a configurable *QPSK* ($M = 2$) or *QAM* ($M = 4$) configuration. Omitted rates equal to 1.

In summary, there are four principal parameters: β, M, N and L, where L depends on the cyclic prefix, N is the OFDM symbol length ($N = 512$ or $N = 1024$) and β, which varies between 1 and 100, is the number of OFDM symbols to be processed in a single activation of the actor. For example, if $M = 4$ and $\beta = 10$, this means that the system is operating in a mode that uses QAM as the demapping scheme, and executes actors in blocks of 10 firings each. A possible scheduling for this application: *SRC* [*CON RCP FFT DUP*

QPSK QAM] *TRAN SNK*. When *SRC* fired, it send a data token to the control node, which decides to choose between *QPSK* or *QAM*, depending on the value of M. The square bracket is used to inform the compiler the region influenced by the control token, starting from the firing of the control node and terminating with the firing of all nodes which receive this control token. Figure 8 presents the minimum buffer size required by the application, depending on the vectorization degree β and the symbol length N ($L = 1$ and M is chosen by the control node). We find out that the buffer size increases proportionally to the vectorization degree and we have an improvement of 29% in comparison with the implementation by using CSDF. This result can be explained by the fact that the dynamic topology obtained using TPDF is more flexible than the static topology of CSDF, allowing to remove unused edges and decrease the minimum buffer size required by one iteration of the TPDF graph. In a similar way, several StreamIt benchmarks (e.g., FM Radio [11]) must perform redundant calculations that are not needed with models allowing dynamic topology changes such as TPDF.

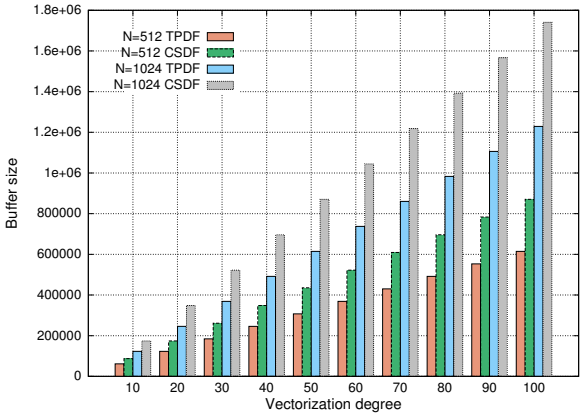


Fig. 8. Minimum buffer size increased proportionally to β, given by $Buff = 3 + \beta \times (12 \times N + L)$ for TPDF and $Buff = \beta \times (17 \times N + L)$ for CSDF.

V. Related work

Several *parametric* dataflow models have been proposed, for instance PSDF [12], VRDF [13], SPDF [5]. In contrast to CSDF, these models allow a dynamic variation of the production and consumption rates of actors to change at runtime according to the manipulated data. However, none of these models provide any of the static guarantees that TPDF does (rate consistency, boundedness and liveness) or propose parametric rates without dynamic topology changes. Another related model is the Scenario-Aware Data-Flow (SADF) MoC [14], which exploits also the mode-based approach where the dynamic behavior of an application is viewed as a collection of different scenarios; yet, our model is more generic as it provides a unified view of manycore systems, which is entirely composable. Our approach is somewhat similar to Boolean Parametric Dataflow (BPDF) [6] which allows not only dynamic variations of rates but also dynamic changes of the graph topology. Each BFDF edge can be annotated with a boolean parameter which changes the topology of the graph. When a boolean parameter is false, the edge annotated by this parameter is considered absent. Still, this is not enough because this model lacks the ability to impose real-time constraints, a feature also required to program modern safety critical applications which will be both highly parallel and time constrained. Our model already solves this problem by using control actors of type clock. Moreover, all SPDF and BPDF case studies (e.g., the VC-1 Video Decoder), presented in [5], [6] and [15], can be replicated using our approach without introducing parameter communication and synchronization between firings of modifiers and users, which would have complicated scheduling significantly because of the additional actors, edges and ports. We also improved the quality of the AVC Encoder, a much more complex application, by using a quality threshold for the motion vector detection, implemented with a Transaction kernel, to choose dynamically the highest quality video available within real-time constraints.

VI. Conclusion

We presented TPDF, a novel parametric dataflow model of computtion for embedded streaming applications. TPDF extends CSDF with parametric rates and a new type of control actor, channel and port, which expresses dynamic changes of the graph topology and time-triggered semantics. We described static analyses to guarantee the boundedness and liveness when executing a TPDF application, and validated these on realistic applications.

References

[1] B. D. de Dinechin, P. G. de Massas, G. Lager, C. Léger, B. Orgogozo, J. Reybert, and T. Strudel, "A distributed run-time environment for the Kalray MPPA-256 integrated manycore processor," *Procedia Computer Science*, 2013.

[2] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," in *Proceedings of the IEEE, vol. 75, no. 9,*, 1987, pp. 1235–1245.

[3] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete, "Cyclo-static data flow," in *ICASSP*, vol. 5, May 1995, pp. 3255–3258.

[4] S. Louise, P. Dubrulle, and T. Goubier, "A model of computation for real-time applications on embedded manycores," in *MCSoC*, Sept 2014.

[5] P. Fradet, A. Girault, and P. Poplavko, "SPDF: A schedulable parametric data-flow MoC," in *DATE*, March 2012, pp. 769–774.

[6] V. Bebelis, P. Fradet, A. Girault, and B. Lavigueur, "BPDF: A statically analyzable dataflow model with integer and boolean parameters," in *EMSOFT*, 2013, pp. 3:1–3:10.

[7] S. S. Battacharyya, E. A. Lee, and P. K. Murthy, *Software Synthesis from Dataflow Graphs*. Kluwer Academic Publishers, 1996.

[8] X. Do, S. Louise, and A. Cohen, "Managing the latency of data-dependent tasks in embedded streaming applications," in *MCSoc*, 2015.

[9] P. Aubry, P.-E. Beaucamps, F. Blanc, B. Bodin, S. Carpov, L. Cudennec, V. David, P. Dore, P. Dubrulle, B. D. de Dinechin, F. Galea, T. Goubier, M. Harrand, S. Jones, J.-D. Lesage, S. Louise, and R. Sirdey, "Extended cyclostatic dataflow program compilation and execution for an integrated manycore processor." in *ICCS*, vol. 18, 2013, pp. 1624–1633.

[10] J.-J. van de Beek, M. Sandell, M. Isaksson, and P. Ola Borjesson, "Low-complex frame synchronization in OFDM systems," in *ICUPC*, 1995.

[11] W. Thies and S. Amarasinghe, "An empirical characterization of stream programs and its implications for language and compiler design," in *Proceedings of PACT*, 2010.

[12] B. Bhattacharya and S. Bhattacharyya, "Parameterized dataflow modeling for DSP systems," *IEEE Transactions on Signal Processing*, 2001.

[13] M. Wiggers, M. Bekooij, and G. Smit, "Buffer capacity computation for throughput constrained streaming applications with data-dependent inter-task communication," in *RTAS*, April 2008, pp. 183–194.

[14] B. Theelen, M. Geilen, T. Basten, J. Voeten, S. Gheorghita, and S. Stuijk, "A scenario-aware data flow model for combined long-run average and worst-case performance analysis," in *MEMOCODE*, 2006, pp. 185–194.

[15] V. Bebelis, P. Fradet, and A. Girault, "A framework to schedule parametric dataflow applications on many-core platforms," in *LCTES*, 2014, pp. 125–134.