VR-PRUNE: Decidable Variable-Rate Dataflow for Signal Processing Systems

Jani Boutellier, *Senior Member, IEEE*, Yujunrong Ma, Jiahao Wu, Mir Khan, and Shuvra S. Bhattacharyya, *Fellow, IEEE*

***Abstract*—The dataflow concept has been successfully used for modeling and synthesizing signal processing applications since decades, and recently, dataflow has also been discovered to match the computation model of machine learning applications, leading to extremely successful dataflow based application design frameworks. One of the most attractive features of dataflow, especially for signal processing, is related to its formal nature: when properly defined, a dataflow-based application model can be analytically verified for correctness at the stage of application design. This paper proposes VR-PRUNE, a novel dataflow model of computation that is aimed for design of high-performance signal processing software, together with runtime support that allows efficient application deployment to heterogeneous GPU-equipped platforms. Compared to prior work, VR-PRUNE features variable token rate processing, which enables designing adaptive signal processing applications, and implementing solutions that, e.g., allow trading-off between power consumption and filtering bandwidth at runtime. The paper presents the formal concepts of VR-PRUNE, as well as four application examples from domains related to signal processing, accompanied with quantitative results, which show that using VR-PRUNE enables, for example, application power-performance scaling, and on the other hand describing adaptive application behavior with 59% fewer dataflow graph components compared to previous work.**

***Index Terms*—Dataflow computing, design automation, signal processing, parallel processing.**

I. Introduction

DATAFLOW modeling for signal processing systems has been investigated actively since the 1980 s. Many widely used signal processing flavored design frameworks employ dataflow concepts — a couple of prominent examples are GNU Radio [2] and TensorFlow [3]. In the application areas of machine learning (Tensorflow) and software-defined radio (GNU Radio), dataflow features several advantages over conventional, unstructured software design approaches: it provides application modularity, software reuse, concurrency and support for heterogeneous computing.

The dataflow concept however exists in multiple Models of Computation (MoC) that have varying features, especially in terms of *analyzability* and *expressiveness*. For a MoC, analyzability refers to the model's predictability: e.g., a well-analyzable model enables a software compiler to reason about the application's execution flow, apply powerful software optimizations and guarantee absence of deadlocks. Expressiveness, in contrast, refers to the model's flexibility in describing the structure or run-time behavior of an application. In many cases, analyzability and expressiveness are contradictory properties of MoCs.

A key aspect of a dataflow MoC that influences both its expressiveness and analyzability is support for conditional execution — in particular, support for decision making that is required for implementing fundamental *if-then-else* and *for*-loop behavior within the dataflow model. Since many classical signal processing applications behave in a very static fashion (in terms of the rates at which functional modules exchange data), fully static dataflow MoCs, such as *synchronous dataflow* (SDF) [4], have been successfully used also in industrial software (e.g. National Instruments LabView [5] or Keysight SystemVue [6]). However, as algorithms in various signal processing domains, such as wireless communications, video coding and machine learning are exhibiting increasing levels of dynamics and configurability, the need for conditional execution at the dataflow level is becoming important for making modern dataflow frameworks sufficiently expressive. Recent examples of this are *adaptive inference graphs* [7] and *hydra nets* [8] that adaptively switch or skip computations in Convolutional Neural Network (CNN) inference.

In a general sense, such adaptive computation scenarios require the underlying dataflow MoC to support conditional execution, which has traditionally been associated with *dynamic dataflow*, such as *Boolean dataflow* (BDF) [9]. However, it has been shown that the general problem of determining whether a BDF graph can be scheduled for execution with bounded memory, is undecidable [1]. Throughout this paper, the following definition of dataflow graph *consistency* is adopted:

Definition 1 (Consistency): A dataflow graph is consistent if it can be scheduled with guarantees of bounded memory and deadlock-free operation, regardless of what inputs are applied.

Manuscript received June 24, 2021; revised November 12, 2021 and January 26, 2022; accepted March 16, 2022. Date of publication March 25, 2022; date of current version April 26, 2022. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Yuan-Hao Huang. This work was supported by the Academy of Finland under Grant 309903 Co-EfNet and the U.S. Army Research Office under Award No. W911NF2110258. *(Corresponding author: Jani Boutellier.)*

Jani Boutellier is with the School of Technology and Innovation, University of Vaasa, 65200 Vaasa, Finland (e-mail: jani.boutellier@uwasa.fi).

Yujunrong Ma and Jiahao Wu are with the Department of Electrical and Computer Engineering, University of Maryland, College Park MD 20742 USA (e-mail: mayu1996@umd.edu; jiahao@terpmail.umd.edu).

Mir Khan is with the Faculty of Information Technology and Communication Sciences, Tampere University, 33014 Tampere, Finland (e-mail: mir.markhan@tuni.fi).

Shuvra S. Bhattacharyya is with the Department of Electrical and Computer Engineering and Institute for Advanced Computer Studies, University of Maryland, College Park MD 20742 USA (e-mail: ssb@umd.edu).

Digital Object Identifier 10.1109/TSP.2022.3162388

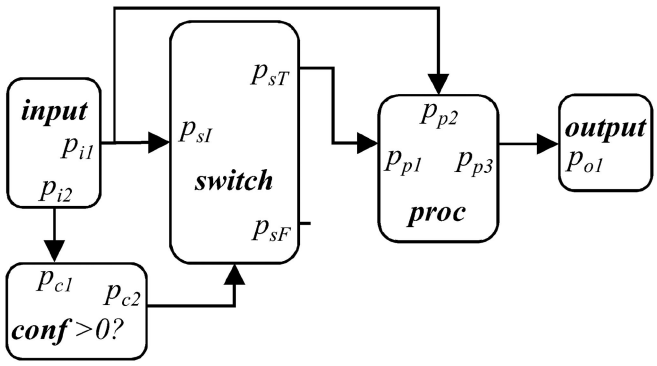


Fig. 1. A weakly consistent dataflow graph. Adapted from [1]. The *conf* actor evaluates sample values coming through port p_{c1}, originating from the *input* actor. *conf* emits Boolean True through port p_{c2} if the sample value is > 0, and False otherwise.

Consider Fig. 1, which shows a dataflow graph with conditional behavior. The *input* vertex produces samples to the three edges that depart from it, whereas the *conf* vertex evaluates whether incoming sample values are greater than zero. In case of a sample value > 0, *conf* emits a Boolean *True* value to its output p_{c2}, otherwise a value of *False*. The *switch* vertex relays the sample coming from p_{sI} to its output p_{sT} if it has received a *True* value from *conf*, otherwise it relays the sample to its output p_{sF}. The graph of Fig. 1 is *weakly consistent* – all samples produced within the graph do not necessarily become consumed, leading to potentially unbounded use of memory: for each sample value ≤ 0 emitted by the *input* vertex, one token is accumulated to the p_{i1} — p_{p2} edge. In other words, unless the *input* vertex doesn't eventually settle into emitting token values > 0, unbounded use of memory will ensue.

It depends on the MoC adopted by the dataflow design framework, how a weakly consistent graph such as the one in Fig. 1 is treated. In a design framework that follows the restricted SDF [4] MoC, the graph could not be modeled at all, as SDF does not allow conditional execution. As another example, in TensorFlow, which is more flexible in this respect, control flow operations [10] would enable implementing the graph, causing version-dependent behavior[1].

The dataflow MoC and design framework VR-PRUNE advocated in this paper addresses the inherent conflict between analyzability and expressiveness in a different way: the VR-PRUNE MoC enables describing conditional execution, however regulated by a set of formal rules and design patterns, which ensure that graph inconsistencies can be detected at design time.

Furthermore, VR-PRUNE features support for variable token rates, which increases the model's expressiveness compared to previous works. Consequently, in this paper we show that VR-PRUNE offers a theoretically solid dataflow basis for future programming frameworks similar to what TensorFlow and GNU Radio are at the moment.

Some of the main ideas of VR-PRUNE were briefly presented in [11] recently. This full-length article extends the conference paper by

- A complete theoretical presentation of the VR-PRUNE Model of Computation,
- Design rules and patterns that ensure compile-time consistency analysis of VR-PRUNE application graphs,
- An open source[2] runtime framework *VPRF* that has been constructed around VR-PRUNE concepts, and
- Run-time experiments on heterogeneous desktop and embedded platforms, which highlight the efficiency and expressiveness of VR-PRUNE.

The rest of this paper is organized as follows: Section II presents related works, Section III describes the proposed VR-PRUNE Model of Computation, Section IV presents the formal VR-PRUNE design rules, Section V discusses consistency of VR-PRUNE graphs, Section VI shows the experimental results related to VR-PRUNE, Section VII discusses the proposed work, and Section VIII concludes the paper.

II. Background

In the dataflow programming concept, applications are expressed as directed *graphs*. The application graph consists of nodes, which are called *actors* that are interconnected by *edges*. Edges carry data that is encapsulated within *tokens*; for example, in an image processing application, a single token may encapsulate the data related to one image pixel, or the data of a complete frame, depending on the application.

The interconnection between an actor and an edge is called a *port*. A port that is connected to an edge, which departs from an actor is called an *output port*, and respectively a port that is connected to an incoming edge is called an *input port* of an actor. Each port is associated with a non-negative integer value called *token rate* that determines how many tokens the actor consumes (for input ports) from its associated edge, or how many tokens the actor produces (for output ports) to its associated edge upon one *firing*.

Firing is the dataflow concept for computation. Consider a simple actor a that has the mere purpose of dividing one integer value with another integer. Logically, a should have two input ports, one for the divisor and one for the dividend, as well as one output port for the result. In order to perform the computations related to a division operation, a needs to have one token available from the port that provides the dividend value, and one token from the port that provides the divisor value. Hence, we can say that for the dividend port a has a fixed input token rate of one, which is also the case for the divisor port and the result port.

Another well-known model for describing information flow in dynamic discrete systems is Petri nets [12]. The main advantage of dataflow graphs over Petri nets is their compactness and convenience in mapping real-life applications to graph-based models [13]. However, a significant class of dataflow models can be transformed to Petri nets, thus enabling application of analytic methods that have been designed for Petri nets, to dataflow models [14].

Most of the differences between existing dataflow MoCs associate with rules and restrictions related to token rates. *Homogeneous* dataflow MoCs require that every port of each actor has

[1] In TensorFlow 1.14 the execution of the graph caused a runtime error, whereas in 2.3.1 the *sink* vertex received 0 samples.

[2] Available at https://gitlab.com/jboutell/vprf

strictly a token rate of one, whereas the synchronous dataflow (SDF) [4] MoC allows a fixed positive integer token rate for each port. Examples of design frameworks that are based on SDF are PREESM [15] and StreamIt [16], [17]. Cyclo-static dataflow (CSDF) [18], on the other hand, enables token rates to change in fixed, periodic cycles. SDF (and its homogeneous variant) and CSDF are regarded as *static* dataflow MoCs, as their token rates are completely predetermined at application design time. Consequently, the dataflow graphs are analyzable at compile-time, enabling formal proofs for absence of deadlocks and bounded memory.

Somewhat more flexibility can be added to the SDF MoC by adding *scenarios*, different SDF graph topologies that are at runtime switched, e.g., based on a finite state machine. The baseline work in this direction is SADF [19], and recently also full software design frameworks such as HOPES+ [20] have been built around the SADF concept.

Dynamic dataflow MoCs, of which *dataflow process networks* (DPN) [21] is one of the most well-known examples, allow port token rates to change arbitrarily at application run time, and therefore possibilities for graph consistency verification at design time are very limited. In the past decade, dynamic dataflow around the CAL language [22] and its subvariant RVC-CAL [23] has triggered the development of several design frameworks such as Tÿcho [24], Orcc [25] and SHeD [26].

Between the extremes of fully static and fully dynamic dataflow exist a number of MoCs that balance between analyzability and expressiveness. *Well-behaved dataflow* [27] restricts dynamic application behavior to take place within subgraphs that follow a predefined topology. These subgraph templates enable expressing conditional constructs such as if-then-else and loops at the dataflow level, while still guaranteeing finite-time verification for bounded memory. Our recent work PRUNE [28] elaborated the ideas of WBDF into a MoC accompanied with a high-performance runtime framework for heterogeneous computing, and design time algorithms for verifying graph consistency.

Another branch of work in boundedly dynamic dataflow is *variable-rate dataflow* (VRDF) [29]. The paper [29] presents a dataflow MoC that allows non-negative integer port token rates that can vary between arbitrary pre-defined limits (called *variable-rate* from here on), and an algorithm that computes the memory capacity required to execute the graph. Additionally, the paper [29] describes a check procedure for determining if a given graph is valid for memory capacity computation. The proposed VR-PRUNE MoC adopts the concept of variable token rates from VRDF, however the emphasis of VR-PRUNE is on high-performance processing on heterogeneous platforms, and consequently VR-PRUNE introduces the restriction of *symmetric token rates*, preventing direct adoption of VRDF models to VR-PRUNE.

Table I summarizes related dataflow MoCs and frameworks detailing their features: *decidable* indicates whether graph consistency analysis is a decidable problem, *dynamic* expresses whether the model supports port token rates that can at run time vary according to a two-valued function, *high-performance* tells whether a design framework with performance metrics has been published, and *variable-rate* shows whether the model permits variable-rate port token rates. Variable-rate dataflow can be considered as a generalization of two-valued dynamic dataflow, and in Section VI we will show that variable-rate dataflow provides a higher degree of expressiveness in terms of describing the same application behavior with considerably fewer dataflow graph elements than two-valued dynamic dataflow.

TABLE I
COMPARISON TO RELATED DATAFLOW MODELS AND LANGUAGES

Work	Decidable	Dynamic	High-performance	Variable-rate
SADF [19]	+	+	-	+(-)
BDF [9]	-	+	-	-
DAL [30]	-	+(-)	+	+(-)
RVC-CAL [25]	-	+	+	+
StreamIt [17]	+	-	+	-
PRUNE [28]	+	+	+	-
VRDF [29]	+	+	-	+
VR-PRUNE	+	+	+	+

Cases requiring explanation in Table I are: SADF [19] – variable-rate dataflow is in principle possible, but this requires one dataflow graph per specific token rate making the solution impractical for larger rate variations; DAL – dynamic and variable-rate dataflow can be built on top of the framework, but not for GPU-accelerated graph components.

The proposed VR-PRUNE Model of Computation builds on concepts introduced in VRDF [29], WBDF [27], and our previous work PRUNE [28]. Consequently, the VR-PRUNE MoC features

- A dataflow model that supports dynamic token rates,
- Token rate variability within pre-defined limits, and
- Design time analysis for bounded memory and absence of deadlocks through a set of design rules and patterns.

In Section VI of this paper we show that token rate variability enables 1) expressing data dependent graphs in a more compact representation, 2) capturing application behavior that was not possible in our previous work PRUNE. Additionally, we show that the increased flexibility of VR-PRUNE does not add computational overhead compared to PRUNE. The following section formally presents the VR-PRUNE MoC.

III. PROPOSED MODEL OF COMPUTATION

In this section, the components of VR-PRUNE graphs are introduced together with an example graph.

A. Notation and Port Types

Following the notation of our previous work [28], a VR-PRUNE application graph $G = (A, F)$ consists of a set of actors A, and a set of directed edges F that interconnect the actors of A. By definition of dataflow, the edges follow first-in-first-out (FIFO) communication behavior, and for this reason we interchangeably also refer to edges as FIFOs. Actors connect to edges over ports, which are classified into *input ports* (for ports that consume tokens) and *output ports* (for ports that produce tokens). Each actor $a \in A$ can contain any non-negative number of input and output ports, and $a = parent(p)$ denotes an actor a contains port p. More briefly, this can also be expressed by p^{+}_{a1}, which refers to the first output port of actor a (Note: the subscript

does not indicate any form of multiplication, only indexing). The superscript $+/-$ corresponds to output/input port respectively.

VR-PRUNE ports consist of three different port types: each port is either a (input or output) *control port*, *static regular port (SRP)* or *dynamic regular port (DRP)*. A SRP p has a fixed token consumption/production rate $atr(p)$, which is set at application design time. A DRP p, however, has a variable token rate that is defined as $lrl(p) \leq atr(p) \leq url(p)$, where non-negative integers lrl, atr and url stand for lower rate limit, active token rate, and upper rate limit of p, respectively. lrl and url are values that are fixed at design time, whereas atr may vary within the limits of lrl and url at run time. Finally, a control port p must have a fixed token rate of 1.

Each FIFO $f \in F$ is connected to exactly one output port p^+ of actor $parent(p^+)$ and to exactly one input port p^- of $parent(p^-)$. Moreover, $fifo(p_a^+)$ and $fifo(p_b^-)$ refer to the FIFO connected to ports p_a^+ and p_b^-, respectively. Ports p_a^+ and p_b^- are *connected* when $fifo(p_a^+) = fifo(p_b^-) = f$, where actors a and b are referred as the source ($source(f)$) and sink ($sink(f)$) of the same FIFO f. In VR-PRUNE, connected ports must always be of the same port type, and a valid VR-PRUNE graph is not allowed to have unconnected ports, as for example the port F of Fig. 1.

In the VR-PRUNE MoC, an output port p^+ that is a control port or SRP, can be connected to multiple FIFOs, but in this case each FIFO must have a unique source and sink port, and every input port p^- must have only one FIFO connected to it. If an output port p^+ is connected to multiple input ports $p_i^-, i = 1 \ldots K$ in the aforementioned way, then each p_i^- must be of the same port type as p^+.

The current number of tokens in FIFO f is denoted as $tokencount(f)$, the FIFO's token capacity is given by $capacity(f)$, and $delay(f)$ denotes the number of delays (initial tokens) in f.

Similar to its predecessor PRUNE, VR-PRUNE adopts the concept of *symmetric-rate dataflow*, which requires for connected ports p_a^+ and p_b^- that $atr(p_a^+) = atr(p_b^-)$. A VR-PRUNE actor a can fire when a) for each input port p_a^- holds $tokencount(fifo(p_a^-)) \geq atr(p_a^-)$, and b) for each output port p_a^+ holds $capacity(fifo(p_a^+)) - tokencount(fifo(p_a^+)) \geq atr(p_a^+)$.

B. Actor Types

Each actor in a VR-PRUNE graph G must fall into one of the four actor types that are characterized by numbers, types and directions of ports, as described below. If an actor does not meet the requirements of any of the four actor types, the actor cannot be included into VR-PRUNE graph G.

1) Static Processing Actor (SPA): The ports of SPA actors can only be of the type SRP, and therefore an SPA actor can be understood to operate similar to an actor of the SDF [31] MoC.

2) Dynamic Actor (DA): A DA has at least one DRP, at least one input control port, and any non-negative number of SRPs. All the DRPs of DA x need to be either of input direction, or of output direction as required by the design rules (Section IV). This restriction enforces modularity of VR-PRUNE graphs and acts as a necessary condition for analyzability. Furthermore, the number of DRPs in x must be greater or equal to the number of its input control ports, since each DRP of x is controlled by exactly one input control port of x.

When a DA x first fires, x first consumes one token from each of its control ports. The values of these consumed tokens set the $atr(p_x)$ for each DRP p_x of x. After the *atr*s of each DRP p_x have been set, firing of x proceeds by following usual dataflow semantics: tokens are consumed from the input ports of x according to the port-specific *atr* values, and consequently tokens are produced to the output ports of x following the port-specific *atr* values.

Fig. 2 shows a VR-PRUNE subgraph with one configuration actor q and two dynamic actors, x and y. The figure illustrates how the token values originating from black-colored input control ports p_{x1} and p_{x2} associate with the DRPs p_{x3}, p_{x4} and p_{x5} in a one-to-many relationship. For actor x that has a DRP p, $cport(p)$ is the input control port of x that is associated with p. Drawing an example from Fig. 2, $cport(p_{x3}) = p_{x2}$.

3) Dynamic Processing Actor (DPA): DPAs are required to have at least one input DRP, at least one output DRP, at least one control input port, and any number of SRPs. In the beginning of firing DPA a, a first consumes one token from each of its control ports. The values of the tokens originating from the control ports set the atr for each DRP of a. Similar to DAs, the number of DRPs must be greater or equal to the number of input control ports, and firing of a proceeds by consuming $atr(p_{ai}^-), i = 1, 2, \ldots, K$ tokens from each of a's K input (SRP or DRP) ports, finally producing $atr(p_{aj}^+), j = 1, 2, \ldots, L$ tokens to each of a's L output (SRP or DRP) ports. Evidently, FIFO *f* that is connected to DRP p must have a token capacity of at least $url(p)$: $capacity(fifo(p)) \geq url(p)$.

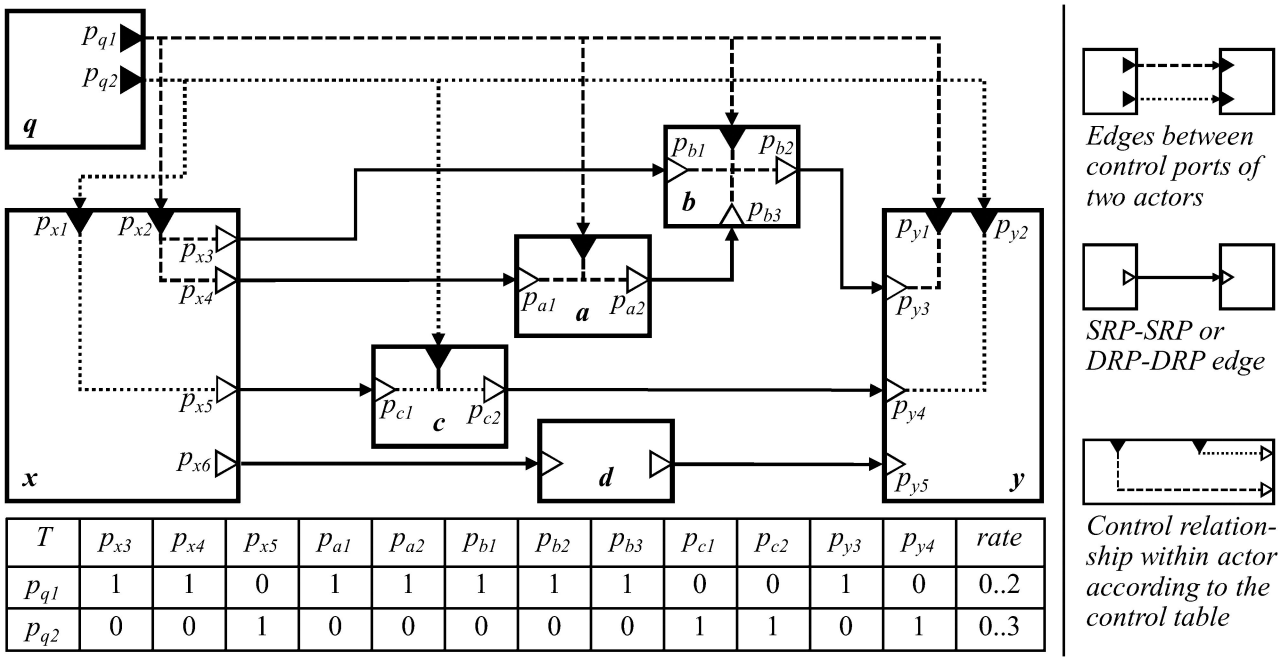
In Fig. 2, a, b and c are DPAs. Out of these, b has two input DRPs (p_{b1} and p_{b3}) and one output DRP (p_{b2}), all of which are controlled by the single input control port of b.

4) Configuration Actors: A configuration actor (example: q in Fig. 2) must have one or more output control ports, which are required, by definition, to have a token rate of unity. Additionally, a configuration actor can have zero or more data ports, which are SRPs. The data ports can either have input or output direction.

The tokens produced by the output control ports contain non-negative integer values that define port-specific atrs for DAs and/or DPAs that consume the control tokens. The relationships between output control ports of configuration actors and input control ports of DAs/DPAs are unambiguously defined by a *control table* that is described below.

C. Control Table and Firing

In the VR-PRUNE MoC, variable token rates are restricted to subgraphs called Dynamic Processing Graphs (DPGs) that define graph-level structure for ensuring analyzability. The VR-PRUNE concept allows various DPG types, however in this paper we define one specific DPG type in Section V, which is suitable for all application examples presented in Section VI. DPG types may impose additional restrictions to actor types, actor port counts, or connectivity between actors.



T	p_{x3}	p_{x4}	p_{x5}	p_{a1}	p_{a2}	p_{b1}	p_{b2}	p_{b3}	p_{c1}	p_{c2}	p_{y3}	p_{y4}	*rate*
p_{q1}	1	1	0	1	1	1	1	1	0	0	1	0	0..2
p_{q2}	0	0	1	0	0	0	0	0	1	1	0	1	0..3

Fig. 2. VR-PRUNE subgraph example. An example of control relationships between a configuration actor (q), dynamic actors (x, y), and DPAs (a, b, c) expressed equivalently by a control table, and graphically. Note: the *rate* column of the control table refers to the range of valid atr values (defined by lrl and url) emitted by the output control port.

The complete VR-PRUNE application graph G can consist of any number of interconnected DPGs: dynamic actors and configuration actors of a DPG are allowed to connect outside the DPG over SRPs without restrictions. In fact, using the concept of hierarchical dataflow graphs (e.g. [32]), a DPG could be represented as a composite actor with static token rate ports. Formal presentation of hierarchical graphs is however limited outside the scope of this work as clustering of actors may change the graph semantics and cause deadlock [32]. In the rest of this paper the analysis concentrates on internal behavior of individual DPGs.

Fig. 2 shows an example of a DPG, where q is a configuration actor, a, b and c are DPAs, x and y are DAs, and d is an SPA. Each DPG is associated with a *control table* T that unambiguously defines 1) the control relationships between output control ports of configuration actors, and DRPs of DAs and DPAs, and 2) sets limits for variable token rates by means of output control port specific lrls and urls.

The control table of Fig. 2 is shown below the DPG — it is a matrix with dimensions $h \times (w+1)$, where h and w equal the number of output control ports and DRPs, respectively. In the control table, a value of '1' indicates a control relationship between the corresponding output control port (row) and DRP (column), whereas a value of '0' indicates that the control port and DRP are not related. Since each DRP is required to be controlled by exactly one output control port of a configuration actor, a valid control table must have a column sum of '1' for each DRP column. The lrl and url values are defined per output control port in the last column of the control table, and apply to all DRPs that are associated ('1') with that row.

Finally, we define the meaning of a *complete cycle* related to a DPG in the spirit of [9]: assuming that a DPG is consistent (as discussed Section V), we define a complete cycle of a DPG as a sequence of actor executions that returns the DPG to its original state. The execution of a complete cycle S of actors of the form $S = q_1, q_2, \ldots, q_m, a_1, a_2, \ldots, a_n$, where the q_i's are the control actors of the DPG, and the a_i's are the SPAs together with the active DAs and DPAs of the DPG, as determined by the execution sequence $q_1, q_2, \ldots, q_m$. Here, an *active* DA or DPA a means that (1) a has input tokens on all control ports, and (2) a is configured by the incoming control tokens so that at least one DRP of a will have a nonzero rate on the next actor firing. To be valid, a complete cycle must also satisfy the condition that there is no FIFO buffer underflow within the edges of the DPG when executing S.

IV. Design Rules

This section presents the VR-PRUNE design rules, which apply to all types of DPGs, providing generic restrictions for supporting dataflow consistency. The approach of using design rules is similar to previous works [27], [28], [32]. Specific types of DPGs may impose further design restrictions beyond VR-PRUNE design rules. Before presenting the rules, some mandatory definitions are provided.

We define two actors, a and b, as *adjacent*, if at least one port of a is connected with at least one port of b.

Definition 2 (Chain): A *chain* is a non-empty sequence $S = (a_1, a_2, \ldots, a_N)$ of actors, such that $\forall i = 1, 2, \ldots, N$, a_i and a_{i+1} are adjacent.

Consequently, chain S connects a_1 and a_N. Furthermore, if all a_i are distinct, then we call S a *simple chain*.

Suppose p_x and p_y are distinct ports of two actors x and y, respectively. We say that p_x and p_y are *linked ports* if (a) $fifo(p_x) = fifo(p_y)$, or (b) there is a simple chain $(x, a_1, \ldots, a_N, y)$ such

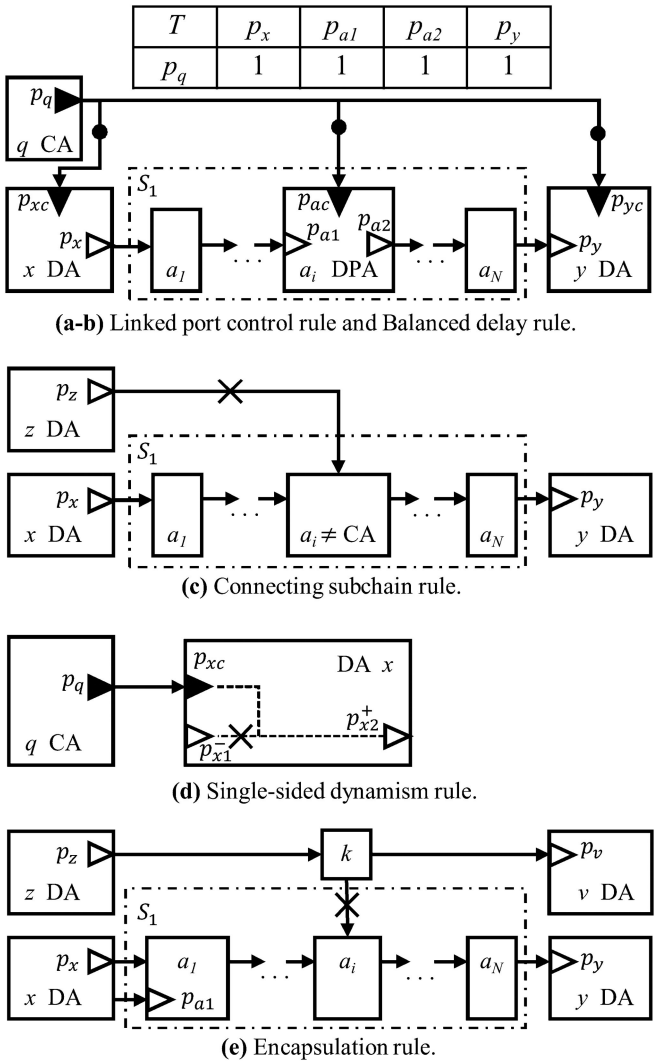


Fig. 3. VR-PRUNE design rules. Illustrations related to design rules (a)–(e). Solid black triangles denote control ports.

that p_x connects with a port of a_1 and p_y connects with a port of a_N. Note that for the linked ports $\{p_x, p_y\}$, there can be multiple *connecting subchains* $a_1, \ldots, a_N$. If p_x and p_y are linked ports, and they are both DRPs, then we say that they are *linked DRPs*.

For all design rules (a)–(e): Suppose $\{p_x, p_y\}$ are linked DRPs whose parents are dynamic actors x and y, and $S_1 = (a_1, a_2, \ldots, a_N)$ is a connecting subchain associated with $\{p_x, p_y\}$:

a) Linked port control rule: Each pair of linked DRPs $\{p_x, p_y\}$, and each DRP within actors of S_1 must be controlled by the same output control port p_q.

b) Balanced delay rule: The input control ports associated with the linked DRPs $\{p_x, p_y\}$, and each DRP within actors of S_1 must be connected to p_q with the same delay. In other words – suppose p_a is a DRP of $a \in S_1$, then $delay(p_q, cport(p_x)) = delay(p_q, cport(p_y)) = delay(p_q, cport(p_a))$.

c) Connecting subchain rule: The actors $a_i, i = 1, 2, \ldots, N$ must all be of type SPA or DPA, and $a_i \in S_1$ may not belong to any connecting subchain $S_2 = (b_1, b_2, \ldots, b_M)$ that is associated with a dynamic actor $z \notin \{x, y\}$.

d) Single-sided dynamism rule: The DRPs of actor x are only allowed to have output direction, and the DRPs of actor y are only allowed to have input direction.

e) Encapsulation rule: Suppose $k \notin S_1$ is an actor that connects to $a_i, i = 1, 2, \ldots, N$. 1) If $k \in \{x, y\}$, then a_i may connect to k only via a DRP p_{a1}. 2) If $k \notin \{x, y\}$, then k must belong to another connecting subchain $S_2 = (b_1, b_2, \ldots, b_M)$ associated with a pair of linked DRPs $\{p_{x2}, p_{y2}\}$ such that $parent(p_{x2}) = x$ and $parent(p_{y2}) = y$.

Fig. 3 illustrates the design rules (a)-(e) as follows: The topmost subfigure is a joint example of rules (a) and (b): the dynamic actors x and y are connected by the chain $S_1 = a_1, \ldots, a_i, \ldots, a_N$, and consequently p_x, p_y and each DRP of each DPA within S_1 need to be controlled by the same control output port p_q of control actor q. Moreover, each FIFO $\{p_q, p_{xc}\}$, $\{p_q, p_{yc}\}$, as well as $\{p_q, cport(p_{a_i})\}$, where p_{a_i} is a DRP of actor $a_i \in S_1$, need to have the same number of delay tokens.

Subfigure (c) of Fig. 3 illustrates the Connecting subchain rule: on one hand, actor a_i of S_1 is not allowed to be a control actor (CA), and on the other hand a_i is not allowed to be part of any connecting subchain that is associated with dynamic actors other that x and y, such as z in the figure.

Violation of the Single-sided dynamism rule (Subfigure d) is illustrated by a dynamic actor x, which incorrectly features both an input DRP p^-_{x1}, and an output DRP p^+_{x2}.

Finally Subfigure (e) of Fig. 3 depicts an actor k that does not belong to the connecting subchain between linked DRPs $\{p_x, p_y\}$. The actor k is part of a connecting subchain that is associated with a different pair of dynamic actors, z and v, which is disallowed. Additionally, Subfigure (e) also illustrates (observe actor a_1) the case where k is one of the two dynamic actors that are interconnected by S_1. In this case, an actor $a_1 \in S_1$ is not allowed to connect to the dynamic actor via a port p_{a1} of type SRP — in contrast, the connection is allowed if p_{a1} is of type DRP.

V. Consistency

This section first defines the Dynamic Processing Graph type used throughout the rest of this paper, and consequently shows that determining its consistency is a decidable problem.

A. The Switch Type Dynamic Processing Graph

The previously introduced VR-PRUNE MoC (Section III) and design rules (Section IV) have been defined without unnecessarily strict constraints, not to limit expressiveness or future developments that build on the MoC. However, for ensuring decidable consistency analysis, additional constraints might need to be incorporated to specific DPG types. Next, we introduce the *Switch DPG* (sDPG) type that is a restricted type of DPG, however generic enough for capturing all the application use cases presented in Section VI. In addition to enabling analysis of the application use cases in our experimental study, the developments we present involving the sDPG demonstrate how groups of relevant applications can be represented and formally analyzed by formulating suitable constraints within the VR-PRUNE modeling framework.

Each sDPG consists of a) exactly one configuration actor q, b) exactly two dynamic actors, x and y, and c) any positive number of *linked DRPs* that connect x and y. These restrictions a)-c) are only associated with the sDPG type, and together with the VR-PRUNE design rules ensure decidable consistency analysis for sDPGs. Other types of DPGs with different restrictions would consequently require a separate consistency analysis procedure.

The linked DRPs of an sDPG establish zero or more connecting subchains between x and y. These chains form the *dynamic components* (DCs) of the sDPG. Given an sDPG D, the set of DCs of D is denoted as $Z_c(D)$, and the pair of dynamic actors in D is denoted $\delta(D) = \{x, y\}$. Fig. 2 shows an example of such an sDPG with actors q, x and y. The DCs of this graph are explained in the end of this subsection.

Consider an sDPG D that contains dynamic actor x with K output DRPs p_{xi} $(i = 1, 2, \ldots, K)$, and dynamic actor y with L input DRPs p_{yj} $(j = 1, 2, \ldots, L)$. We require that each p_{xi} is a linked DRP with at least one of p_{yj}. Our procedure for finding the DCs $Z_c(D)$ associated with a given DPG D can be expressed as follows:

1) For each SRP p_a of actor $a \in \{q, x, y\}$, remove *fifo*(p_a). Next, remove all actors and edges that have become disconnected from the set of actors $\{q, x, y\}$ through the preceding removal of FIFOs *fifo*(p_a).

2) For each linked DRP $\{p_{xi},\ p_{yj}\}$, where *fifo*$(p_{xi}) =$ *fifo*(p_{yi}), insert a dummy actor d (DPA) such that *fifo*$(p_{xi}) =$ *fifo*(p_d^-) and *fifo*$(p_d^+) =$ *fifo*(p_{yj}).

3) Remove q, x, y, and all FIFOs *fifo*(p_q), *fifo*(p_x) and *fifo*(p_y) in D. This removal procedure decomposes D into a set of connected components that form the DCs. Thus, $Z_c(D) = \{Z_1, Z_2, \ldots, Z_M\}$, where $M \in [1, min(K, L)]$ is an integer constant.

For an sDPG D to be *valid*, 1) each DPA a within each $Z_k, k \in [1, M]$ of D must have exactly one control port p_a, and 2) within D, no *fifo*(p), where p is a DRP, may have $delay($*fifo*$(p)) > 0$.

Fig. 2 depicts an example of a valid sDPG D. Following the above described three-stage procedure for discovering DCs, the resulting DCs are $Z_c(D) = \{Z_1, Z_2\}$. The actors related to the DCs are all DPAs, such that $Z_1 = \{a, b\}$, $Z_2 = \{c\}$. Notice that the SPA actor d was removed in the 1st stage, and is not part of the DCs. As required by the design rules (Section IV) and sDPG validity requirements, both DPAs a and b of Z_1 have one input control port each, which is connected to p_{q1}. The control port of actor c (which belongs to DC Z_2) is connected to p_{q2}, and consequently the token rates of ports related to actors within Z_1 and Z_2 can vary independently of each other.

B. VR-PRUNE Graph Consistency

VR-PRUNE graphs may consists of several DPGs, but our design rules ensure that the individual DPGs are independent of each other. Since the existence of DRPs (and hence non-static token rates) is limited to within DPGs, a) the actors outside DPGs and b) ports of DPG actors connecting outside the DPGs necessarily have static token rates. Hence, the consistency of the VR-PRUNE application graph G can be validated using standard SDF validation techniques [31]. Therefore, we limit our discussion on consistency to within sDPGs.

In the following, a proof for the decidability of the consistency analysis of sDPGs is presented. The reasoning followed by the proof is to show that 1) the configuration actor of the sDPG fixes the token rate of each DRP within the sDPG for the duration of one complete cycle of an sDPG, 2) consequently, each DC of the sDPG can be interpreted as a fixed token rate (SDF) graph for the duration of that complete cycle, and 3) finally, the whole sDPG can be considered as an SDF graph, for which it is well-known [4] that determining consistency is a decidable problem.

Lemma 1: Assuming all DAs and DPAs are contained within sDPGs: if all sDPGs of a VR-PRUNE graph G are consistent, then the whole VR-PRUNE graph G is consistent.

Proof: Let $Z_c(D) = Z_1, Z_2, \ldots, Z_M$ be the set of DCs of a valid sDPG D. The actors of DCs are by the Connecting subchain rule required to be of type DPA or SPA.

Since we only consider valid sDPGs, each DRP of $actors(Z_k)$ within a single DC $Z_k, k \in [1, M]$ is controlled by the same output control port p_q of configuration actor q. Consequently, p_q sets the atr of all DRPs within $actors(Z_k)$ to the same value for each complete cycle of an sDPG, and Z_k can be considered as an SDF graph. Since there is a finite maximum of $url - lrl + 1$ different token rates per DC, and, considering that the set of DCs within one sDPG is finite, it is decidable to determine whether or not the sDPG is consistent.

If all the Z_k's are consistent, then there exists a valid, periodic schedule $P(Z_k)$ for each Z_k [4]. $P(Z_k)$ defines a schedule for each actor, $actors(Z_k)$, related to the current atr set by p_q that is associated with Z_k.

Considering $actors(Z_k)$, for each FIFO f connected an actor $a \in actors(Z_k)$, there exists a buffer bound $B_k(f)$ that indicates the maximum token count on f at any stage of $P(Z_k)$ execution. This buffer bound exists as a consequence of SDF graph consistency [4]. Among all DCs Z_c, there is a finite FIFO-specific bound $\beta(f) = max(B_k(f) \mid Z_k \in Z_c(D))$.

The whole sDPG can then be executed by a sequence of schedules $\Omega = (O_1, O_2, \ldots)$ such that for every O_k, there is an $H_k \in Z_c(D)$, where $O_k = P(H_k)$. H_k is the kth executed DC within the sDPG.

As each Z_k is assumed to have a valid, periodic schedule, execution of O_k does not cause a net change to the token counts of the FIFOs between $actors(Z_k)$. Therefore, the token count of FIFO f is bounded by $B_k(f)$. Finally, the token count of f is during execution of Ω is limited to $\beta(f)$. □

VI. Experimental Results

Experiments related to VR-PRUNE were performed on four application use cases: 1) adaptive digital predistortion, 2) parallel image classification, 3) dynamic-update digital predistortion and 4) object detection. Examples 1 and 3 concern real-time signal processing for wireless communications, whereas examples 2 and 4 concentrate on deep convolutional neural networks.

The experimental results show that the increased expressiveness of VR-PRUNE (compared to PRUNE [28])

- Enables expressing the same application structure by clearly fewer dataflow graph elements,
- Enables describing dataflow behavior that cannot be captured by PRUNE,

TABLE II
PLATFORMS USED FOR EXPERIMENTS

Tag	CPU	GPU	Operating System
i7	Intel Core i7-8650U: 1.9 GHz, 4(8) cores	Intel UHD Graphics 620	Ubuntu Linux 18.04
N2	Amlogic S922X (1.8 GHz, 4×ARM Cortex-A73 + 2×ARM Cortex-A53 cores)	ARM Mali G-52	Ubuntu Linux 18.04
XU3	Samsung Exynos 5422 (2.1 GHz, 4×ARM Cortex-A15 + 4×ARM Cortex-A8 cores)	ARM Mali T628	Ubuntu Linux 14.04

- Provides means for saving power by adaptive reduction of computational effort, and
- Does not cause excessive computational run-time overhead.

The first point listed above results because VR-PRUNE enables more compact representation of design functionality. For elaboration on the importance of using compact representations in system-level modeling, see for example [33]. The run-time measurements have been performed on three platforms (see Table II), one of which is a regular mobile workstation (i7), and the other ones are embedded platforms (XU3 and N2), all equipped with GPUs.

A. Run-Time Framework and Application Programming

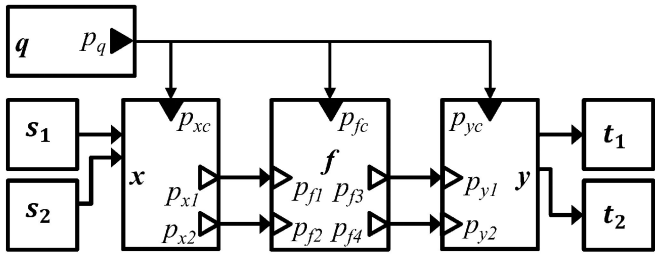
In order to conduct run-time experiments for measuring the efficiency of VR-PRUNE, the PRUNE runtime framework was extended to accommodate variable token rates. We refer to the resulting extension as the VR-PRUNE Runtime Framework (VPRF).

VPRF operates under Linux, and bases its concurrent computation infrastructure on Linux parallel computing primitives: each actor is instantiated as a separate thread that can either be assigned to a specific processor core, or be auto-assigned to a free core by the operating system. FIFOs are implemented as memory arrays, and read/write access to them is arbitrated by *mutex* constructs. Although Section V discussed execution schedules of VR-PRUNE graphs for the purpose of consistency analysis, VPRF follows static assignment scheduling [34] where the operating system determines the execution order of actors at run-time, subject to dataflow constraints.

VPRF also features deeply embedded support for interfacing GPUs. This means that the FIFO and actor primitives of VPRF have been designed from the beginning for efficient data transfers between CPU cores and the GPU, as well as data exchange between GPU kernels, including functionality for variable-length data transfer to/from GPU. VPRF also includes a prototype graph validity checker that inspects application graphs against VR-PRUNE design rules. The dataflow models and experiments described in Sections VI-B and VI-C have been done under VPRF.

The VPRF application programmer writes the application code using the C language or OpenCL for actor descriptions, and XML for describing the application graph. VPRF follows a predefined actor template that originates from the DAL [35] framework: each actor has *initialize*, *fire* and *finish* functions, as well as a persistent actor-specific data structure for preserving actor state between firings. For actors that are aimed to GPU execution, the actor behavior is expressed in the OpenCL language (OpenCL was selected over CUDA for reasons of wider hardware support, as almost all CUDA devices support OpenCL, but not vice-versa). VPRF provides a compact set of

(a) VR-PRUNE graph of adaptive digital predistortion.



T	p_{x1}	p_{x2}	p_{f1}	p_{f2}	p_{f3}	p_{f4}	p_{y1}	p_{y2}	rate
p_q	1	1	1	1	1	1	1	1	1..5

(b) PRUNE graph of adaptive digital predistortion.

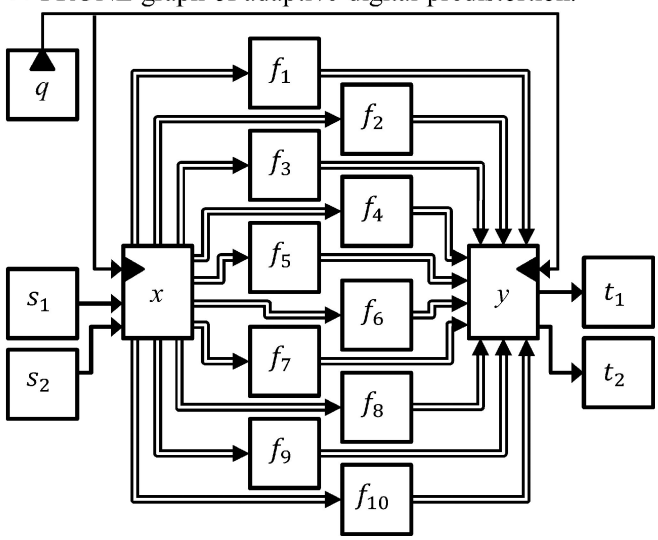


Fig. 4. The adaptive digital predistortion application. Subfigures: (a) VR-PRUNE graph, (b) PRUNE graph. In Subfig. (b) double-line arrows depict 2×FIFOs.

function calls for inter-actor data exchange over FIFO buffers. The function calls effectively hide the complexity related to GPU programming from the programmer: data transfer from a CPU-based actor to a GPU-based actor is carried out using the same function as data transfer between two CPU-based actors. VPRF includes a compiler that transforms the application graph XML file into a top-level C file, which can together with the actor descriptions be compiled into an executable.

Exploiting target platform heterogeneity and parallelism is an important factor for efficient signal processing. Here, VPRF follows the approach first introduced by DAL [35], and later used in PRUNE: using a *mapping specification* (XML file) the application programmer can assign each actor to a specific CPU core, or to the system's GPU. If an actor is assigned to the GPU,

the programmer also needs to indicate the number of *work items* and *work groups* as required by OpenCL [36]. If the work size of an actor exceeds the maximum work size of the GPU, the underlying OpenCL driver automatically divides the processing into multiple passes. On the other hand, if the actor work size is less than the GPU's maximum work size, the GPU runs with reduced utilization; simultaneous execution of several actors on the same GPU is not supported.

CPU core assignments are handled by Linux CPU affinity masks that define which CPUs/cores are eligible to execute each thread (actor). Mapping an actor to the GPU requires expressing the actor's internal behavior in OpenCL, but data exchange between the GPU and CPU, as well as GPU initialization are handled by VPRF.

TABLE III
ADAPTIVE DIGITAL PREDISTORTION LINES OF CODE PER ACTOR

Framework	s	q	x	f	y	t	mean
PRUNE	59	64	139	44	141	48	**83**
VR-PRUNE	59	68	103	64	106	48	**75**

TABLE IV
ADAPTIVE DIGITAL PREDISTORTION THROUGHPUT IN COMPLEX FLOAT MEGASAMPLES/S, AS A FUNCTION OF ENABLED FILTER BRANCHES (%), ON THE I7 CPU AND ON THE N2 CPU. HIGHER IS BETTER

Framework	20%	40%	60%	80%	100%
PRUNE i7	**90.77**	45.75	30.50	22.60	18.29
VPRF i7 (prop.)	88.44	**46.86**	**31.62**	**23.63**	**18.68**
PRUNE N2	17.88	9.53	6.50	4.88	4.11
VPRF N2 (prop.)	**18.25**	**9.62**	**6.77**	**5.20**	**4.29**

B. Expressiveness and Computational Efficiency

One of the main advantages of VR-PRUNE over PRUNE is the increased expressiveness offered by variable token rates compared to binary (on/off) token rates of PRUNE. The increased expressiveness can be quantified in terms of graph size: with a more expressive model it is possible to represent the same functionality with fewer elements (actors, edges) than with a less expressive model. The *adaptive digital predistortion* and *parallel image classification* application examples that are introduced next highlight that in addition to expressiveness, variable token rates provide potential for saving power by computational effort reduction, while maintaining implementation efficiency.

1) Adaptive Digital Predistortion: Digital predistortion is a signal processing approach for compensating non-linear effects of a wireless transmitter's power amplifier [37]. The signal processing for predistortion is computationally very demanding, and therefore solutions [38] that trade-off predistortion bandwidth against computational effort are useful for saving power when possible, e.g. due to varying interference conditions.

Fig. 4 shows such an adaptive predistortion filter in two implementations: a) is a VR-PRUNE implementation, and b) is the conventional PRUNE implementation introduced in [28]. Both implementations describe the same 10-tap parallel Hammerstein filter structure [39], with adaptive functionality such that individual filter branches can be enabled or disabled on-the-fly at run time. In Fig. 4, s_1 and s_2 are source actors that provide samples from the transmitter baseband side; x is a dynamic actor that computes polynomial basis functions and distributes the samples to parallel FIR filter branches f; y is a dynamic actor that implements a summation function for combining the filter branch outputs and compensates for I/Q imbalance; t_1 and t_2 finally act as output actors towards the power amplifier. Adaptive processing is controlled by the configuration actor q that based on external input can enable/disable filter branches $f_1 \ldots f_4$ and $f_7 \ldots f_{10}$ in the PRUNE implementation. In the VR-PRUNE implementation the same effect is accomplished by variable token rates to/from the actor f. In the PRUNE implementation the FIR filter actors are SPAs, whereas the VR-PRUNE FIR actor f is a DPA.

It can be seen that the VR-PRUNE implementation (Fig. 4(a) compactly captures the adaptive predistortion functionality within 8 actors and 11 FIFOs, whereas the PRUNE implementation (Fig. 4(b) requires 17 actors and 46 FIFOs (double-line arrows stand for 2×FIFOs, one for the I channel, and one for the Q channel). This reduction of graph elements comes from the fact that VR-PRUNE is able to capture the functionality of actors $f_1 \ldots f_{10}$ within a single actor f, and consequently reduces the number of connecting FIFOs from 40 down to 4. Based on the number of architectural elements, VR-PRUNE therefore reduces the predistortion model complexity by 70% compared to PRUNE. This reduction ratio depends on the structure of the original dataflow graph, more specifically on the count of f (filter) actors.

One potential fallacy in evaluating model complexity reduction as described above is that actor and edge counts can often be reduced by just lumping actors together. For example, an entire SDF graph can be implemented as a single actor, thereby reducing the graph size to 1 actor and 0 edges. To show that this is not the case here, Table III shows the lines of code per actor for the Fig. 4 graphs. It can be seen, that on average, the VPRF implementation requires fewer lines per code than the PRUNE implementation. Looking at the dynamic actors x and y, it can be seen that the PRUNE implementations of these actors have around 30% more code than their VPRF counterparts, which is related to inter-actor communication: initiating and terminating inter-actor data transfer requires two lines of code per FIFO. Since the PRUNE x and y actors have 20 FIFOs towards the f actor each, whereas the VR-PRUNE x and y have only 2, the difference is obvious. In contrast, the VPRF f actor has 20 lines of code more than the PRUNE counterpart – this is due to the fact that the single f actor stores all the filter coefficients that in the PRUNE implementation are distributed over $f_1 \ldots f_{10}$.

Finally, Table IV shows the throughput of the adaptive digital predistortion application for both the i7 and N2 platforms. The throughput is largely limited by the compute resources of each platform, but the runtime framework's impact can still be seen in the performance figures: due to the considerably simpler graph structure, the VPRF implementation (Fig. 4(a) causes reduced computational overhead compared to the conventional PRUNE implementation (Fig. 4(b). It is worthwhile to point out that reducing the number of actors as between Fig. 4(b) and Fig. 4(a) evidently decreases potential for parallel execution. Therefore, if the underlying execution platform has spare cores, it is not advisable to collapse all actors $f_1 \ldots f_{10}$ into a single actor, but instead consider distributing the computation effort evenly over available compute resources.

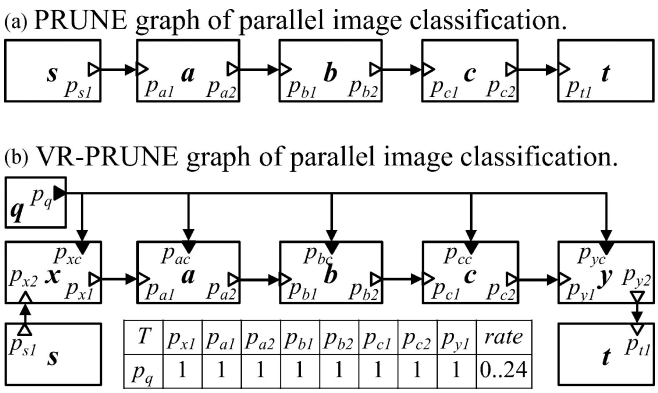


Fig. 5. The parallel image classification application. Subfigure (a) shows the static PRUNE graph, whereas (b) shows the adaptive VR-PRUNE graph. In both graphs, actors a, b, and c are executed on the GPU, and $N = 24$ images are classified in parallel.

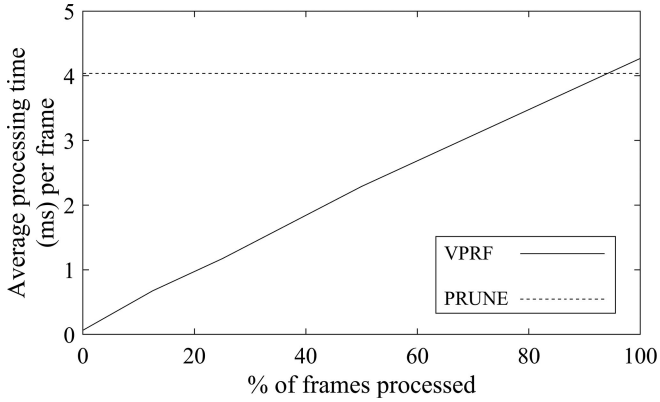


Fig. 6. Parallel image classification performance on the i7 GPU platform for $N = 24$ parallel images: static processing pipeline of PRUNE vs. VPRF adaptive processing by variable token rates.

2) Parallel Image Classification: Image classification is a fundamental computer vision task that is nowadays exclusively performed by deep CNNs. Driven for example by surveillance applications [40], there has also been interest in pushing classification to be done on low-resource computing devices. Some edge/IoT computing platforms such as the Odroid XU3 (Table II) are also equipped with a reasonably powerful GPU that offers optimal performance and efficiency only when the processing workload is sufficiently parallel.

Fig. 5(a) shows a PRUNE graph for parallel image classification: actor s is the source actor that acquires $N = 24$ parallel images from a source (e.g., camera interface), and using the underlying PRUNE CPU-GPU interface sends the N 96 $\times$ 96 RGB images in parallel to the GPU-mapped actor a that performs 2D convolution, ReLU non-linearity, max-pooling and $2\times$ subsampling to all N input images in parallel. Actor b (on GPU) also performs 2D convolution, max-pooling and subsampling, followed by the GPU-mapped actor c that implements a dense layer. After actor c, the feature maps of the N images are transferred back to CPU processing by the PRUNE infrastructure for final processing by two small dense layers, ReLU activations and softmax output, all performed by actor t. All actors in the Fig. 5(a) graph are SPAs.

Fig. 5(b) shows the VR-PRUNE graph for the same CNN-based image classifier, however modified such that the configuration actor q can at runtime determine which of the N parallel images will undergo classification, and which are skipped. For this, the VR-PRUNE graph contains additional dynamic actors x and y (a, b and c are DPAs in Fig. 5(b).

Fig. 6 shows a useful effect of this adaptivity: the computation effort decreases almost linearly on the i7 platform from the maximum of $N = 24$ processed images down to 0, offering great potential for saving processing power. Fig. 7 shows the similar effect on the XU3 embedded platform for $N = 4$ images.

Fig. 8 shows the power scaling effect of VPRF parallel image classification on the embedded XU3 platform. Summing the power dissipation of the GPU, CPU, and memory, it can be seen that when all frames (100%) are classified, the total power dissipation by the application is 3.4 W. Decreasing the number of classified frames down to 0% by adjusting the token rate, makes the application power dissipation as small as 20 mW. For each hardware component (GPU, CPU, memory), the power figures were acquired from the on-board current sensors of the XU3 platform.

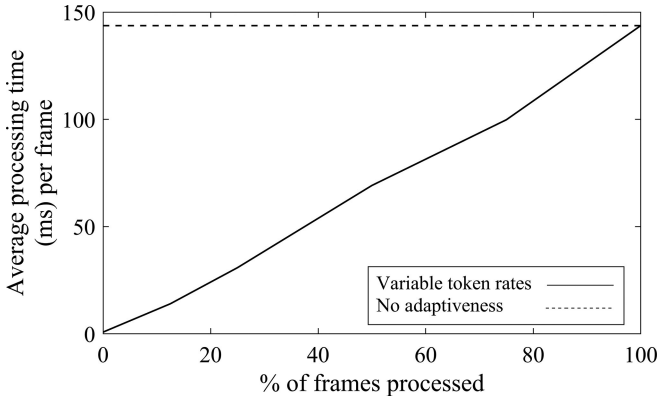


Fig. 7. Parallel image classification performance on the XU3 GPU platform for $N = 4$ parallel images. VPRF adaptive skipping of frames by variable token rates vs. no adaptiveness.

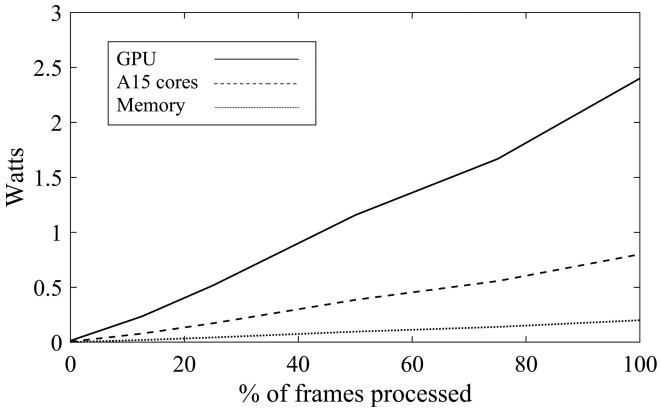
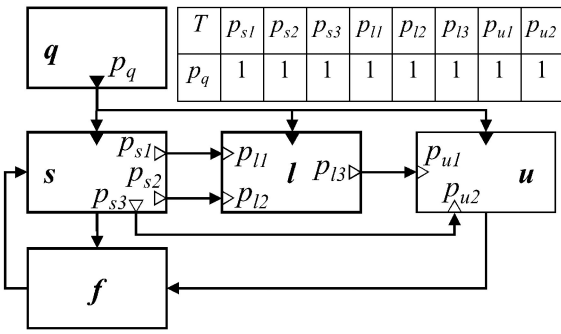


Fig. 8. Parallel image classification power dissipation on the XU3 platform under VPRF for $N = 4$ parallel images. The x axis depicts percentage of frames processed (e.g. 25% = on average every fourth frame undergoes classification).

(a) VR-PRUNE graph of the DU-DPD filter.



T	p_{s1}	p_{s2}	p_{s3}	p_{l1}	p_{l2}	p_{l3}	p_{u1}	p_{u2}	rate
p_q	1	1	1	1	1	1	1	1	$[\frac{N}{2}, N]$

(b) PRUNE graph of the DU-DPD filter.

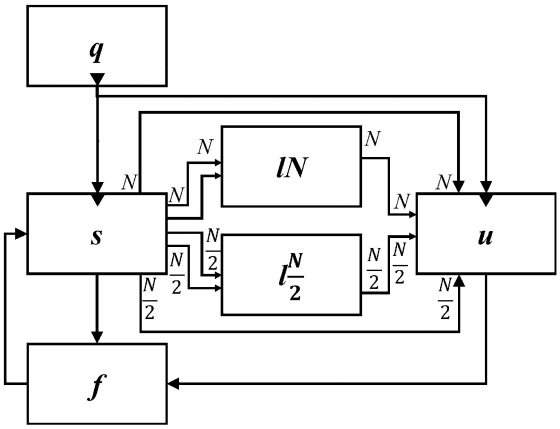


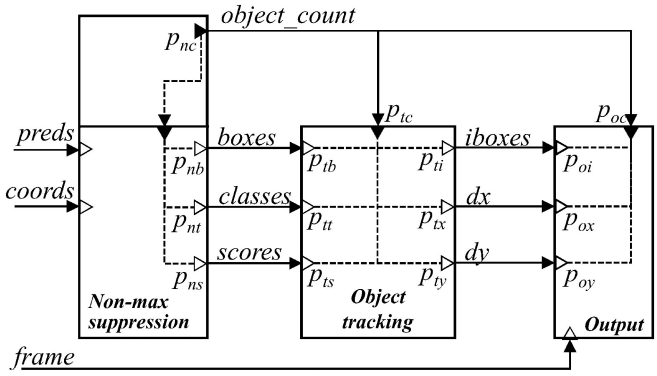
Fig. 9. The dynamic-update predistortion (DU-DPD) filter application. Subfigures: (a) VR-PRUNE and (b) PRUNE graphs. Both graphs implement two alternative token rates: N and $\frac{N}{2}$.

C. Further Application Examples

1) Dynamic-Update Predistortion Filter: The GPU-based dynamic-update predistortion filter (DU-DPD) for 5 G small cells is based on a recent architecture presented in [37]. The original dataflow implementation of [37] was imported to VPRF and modified such that the sample rate of the learning part can be changed adaptively at run-time for the purpose of saving computation effort in situations where radio frequency interference is low. The VR-PRUNE graph of the DU-DPD is shown in Fig. 9(a); the actor *s* contains essentially a power amplifier model, *l* encapsulates decorrelation-based filter coefficient learning functionality, whereas the actor *u* updates filter coefficients. Actual signal predistortion is performed by the *f* actor that has a constant I/O sample rate of N and is executed on the GPU. In practice, N can be e.g. 10000 or 65535 [37].

The variable-rate processing feature of the DU-DPD is exhibited between the actors *s*, *l* and *u:* the current sample rate (atr) can be adaptively changed among $\frac{N}{2}$ and N, as dictated by the *q* actor. The subset of the four aforementioned actors also form the sDPG for the DU-DPD application: *s* and *u* are the pair of dynamic actors δ for the sDPG, and *l* is a DPA. The actor *f* has static token rates at all its inputs and outputs, and hence it is an SPA.

With respect to computational characteristics, DU-DPD differs from the adaptive digital predistortion application of Section VI-B in two ways: 1) DU-DPD includes a feedback loop,



T	p_{nb}	p_{nt}	p_{ns}	p_{tb}	p_{tt}	p_{ts}	p_{ti}	p_{tx}	p_{ty}	p_{oi}	p_{ox}	p_{oy}	rate
p_{nc}	1	1	1	1	1	1	1	1	1	1	1	1	0..9

Fig. 10. The object detection and tracking application. VR-PRUNE graph of the SSD-Mobilenets object tracking component. The full application graph consists of 53 actors and 72 edges.

and 2) the learning algorithm (actor *l*) is recursive by nature. Here, recursiveness means that the algorithm operates on blocks of samples, and with respect to the algorithm's output, sample value s_{i+1} depends on sample value s_i ($i \in [0, N-1]$). Therefore, output sample s_{i+1} cannot be computed independently of sample s_i, for all $i \in [0, N-1]$.

The significantly higher expressiveness of VR-PRUNE over PRUNE is illustrated in Fig. 9(a) and Fig. 9(b). Using the PRUNE MoC, implementing several learning rates would require replicating the *l* actor for each different token rate; Fig. 9(b) shows an example, where the *l* actor can operate at sample rates N and $\frac{N}{2}$. For a finer-grained set of token rates of, e.g., 8 different sample rates, the *l* actor and its associated edges would need to be replicated 8 times. In contrast, the VR-PRUNE model is capable of supporting any number of integer sample rates with the simple graph that is shown in Fig. 9(a): the only change required is the token rate range (or rate list) in the control table.

2) Object Detection and Tracking: Visual object detection has been one of the most successful applications of computer vision since the introduction of deep CNNs. A CNN-based object detector can be understood to consist of a few main components: 1) A CNN-based feature extractor, 2) the actual object detector, and 3) object post-processing for application specific purposes. One of the most apparent post-processing operations for object detection is object tracking that can be used to discover the motion trajectories of objects and for maintaining object identifiers across sequences of images.

For VPRF, an object tracker was implemented based on the MobileNets [41] CNN for feature extraction, followed by the SSD (single-shot detector) [42] object detector, and an object tracking block.[3] Fig. 10 depicts the final stages of this 53-actor VR-PRUNE graph: the Non-max suppression (NMS) actor receives object predictions and coordinates from the feature detector CNN, resolving these into the number of detected objects (*object_count*). This number is distributed from the

[3]This MobileNet-SSD implementation is fully stand-alone and requires no external computer vision frameworks or libraries. The trained weights for CNN layers were extracted from TensorFlow using a prototype software tool written by one of the paper authors, M. Khan.

output control port p_{nc} to the Object tracking actor (DPA) and to the Output actor (DA) that performs visualization. The number of detected objects varies frame-by-frame between 0 and MAX_OBJECTS, which is a compile-time constant. Consequently, the number of detected objects directly controls how many bounding boxes (*boxes*), class predictions (*classes*) and prediction scores (*scores*) are communicated from NMS to the object tracking actor. Similarly, this number also adjusts the number of tracked boxes (*iboxes*), and motion vectors (*dx* and *dy*) between the Object tracking and Output actors. As Fig. 10 shows, the NMS actor is a joint configuration and dynamic actor, and hence the pair of dynamic actors $\delta(D)$ is {NMS, Output}. Such a merged actor can be beneficial for, e.g., simplifying the actor network [28]. For the purpose of consistency analysis, as described in Section V, the configuration and dynamic actor should be treated as separate actors, however.

Similar to the DU-DPD application example, implementing the varying object count feature of Fig. 10 without the variable token rate feature of VR-PRUNE would be complicated. The VR-PRUNE graph displayed in Fig. 10 shows an implementation where the tracked object count can vary between 0 and 9 (visible within the control table) — implementing this rate scalability feature within PRUNE would require 9 instances of the object tracking actor.

D. Comparison With Other Frameworks

For years, efficient computing has been of highest importance in signal processing and machine learning, especially in the embedded systems context. Unfortunately, achieving highest computation efficiency requires adapting software to the intricacies of the hardware architecture, which is a very work-intensive task. To this extent, computing hardware manufacturers active in the machine learning domain (NVidia, Intel, ARM, Qualcomm, etc.) have in the recent years released proprietary libraries and frameworks for accelerating machine learning inference on their computing architectures.

The performance of a dataflow flavored runtime framework consists of two components: 1) the actor implementations, and 2) inter-actor communication and synchronization. The scope of the proposed VPRF framework is entirely related to the latter (2) item, whereas towards actor implementations (1) VPRF is implementation-agnostic. To provide perspective over VPRF efficiency, this section shows some results for VPRF in the context of commercial frameworks that utilize optimized actor implementations (1).

In order to benchmark VPRF synchronization efficiency (2) meaningfully, we chose to adopt actor implementations (1) from ARMCL and oneDNN libraries for machine learning on ARM and Intel platforms, respectively. As the programming interface of ARMCL is not directly compatible with VPRF, the experiment required manual program adjustments to the VPRF implementation after automatic VPRF code generation.

Table V provides a performance evaluation on the Image classification application (Fig. 5(a), reflecting the performance of VPRF against TensorFlow, and the optimized actor implementations of ARMCL and oneDNN. The comparison between VPRF and PRUNE is omitted here, because with this graph the runtime computations for VPRF and PRUNE are identical. The first column of the table shows results of an experiment, where image classification has been performed by VPRF and TensorFlow[4] on the i7 CPU, both leveraging oneDNN. It can be seen that VPRF provides a substantially higher processing performance than TensorFlow, related to the fact that VPRF programs are compiled and optimized by the GNU C compiler, whereas TensorFlow emphasizes ease of programming by Python. The last row of column 1 shows for reference the execution time of the same program implemented as single-threaded C code, leveraging the same computation kernels as the VPRF implementation. VPRF distributes the computations across the different cores of the i7 platform, providing higher throughput than the Baseline implementation. For reference, the second row shows a performance figure from Fig. 6 where $N = 24$ images are classified in parallel on the i7's GPU using generic OpenCL actor implementations. It can be seen that the performance-optimized oneDNN CPU actor implementations outperform GPU acceleration in this case. Since TensorFlow requires a CUDA compatible GPU (and TensorFlow Lite requires Android or IOS for OpenCL), the column 2 experiment could not be done for the TensorFlow framework.

TABLE V
IMAGE CLASSIFICATION PERFORMANCE IN THE CONTEXT OF COMMERCIAL FRAMEWORKS. EACH CELL REFLECTS PROCESSING TIME IN MILLISECONDS PER FRAME FOR N IMAGES PROCESSED IN PARALLEL USING THE GRAPH OF FIG. 5(A), AND 100% OF IMAGES UNDERGO CLASSIFICATION

Framework	i7 CPU $N = 1$, oneDNN	i7 GPU $N = 24$	N2 CPU $N = 1$	N2 GPU ARMCL
VPRF	2.1	4.04	153.5	16.8
TensorFlow	13.9	n/a	236.9	n/a
Baseline C	2.9	n/a	n/a	16.6

The last two columns show results using the embedded N2 platform with and without use of the GPU. In the last column, actor implementations were adopted from the ARMCL library and executed on the GPU from a) the Baseline C language program, and b) VPRF, yielding almost identical performance. Since all the significant actors of the Image classification application are executed on the single GPU of the system, there are no possibilities to leverage concurrent computing, and consequently both the Baseline C and VPRF versions effectively execute the classification sequentially. *However, the result shows that the concurrent thread-based VPRF runtime adds negligible overhead compared to the ARMCL accelerated Baseline C implementation.* To achieve highest performance, *GPU-mapped FIFOs* were used in VPRF. Details on this technique are explained in Appendix A. Column 3 also shows a comparison between VPRF with generic actor implementations against TensorFlow[5] CPU on the N2 ARM platform, indicating faster execution for VPRF.

Finally, Table VI shows a detailed comparison of VR-PRUNE vs. other dataflow models in terms of graph sizes. The table shows that for each application VR-PRUNE clearly outperforms both PRUNE and SADF [19] models in expressiveness. VRDF [29] and Dataflow Process Networks (DPN) [21], on the other hand, are more expressive than VR-PRUNE and can consequently capture VR-PRUNE graphs with similar, but a slightly fewer number of components. However, the DPN model

[4]Python 3.6.9, TensorFlow 2.3.1 with eager execution and JiT compilation enabled.

[5]Python 3.5.7, TensorFlow 1.14

TABLE VI
GRAPH SIZE VERSUS OTHER DATAFLOW FRAMEWORKS. THE *RATE* ROW EXPRESSES TOKEN RATE VARIATION FOR EACH APPLICATION, E.G. [0..24] EQUALS TO 25 DIFFERENT TOKEN RATES, AND "$\frac{N}{2}$ OR N" EXPRESSES TWO ALTERNATIVE RATES. FOR EACH FRAMEWORK AND APPLICATION, THE CELLS DENOTE NUMBER-OF-VERTICES, NUMBER-OF-EDGES. FOR EXAMPLE: (8, 11): 8 VERTICES AND 11 EDGES

Application	Adaptive DPD	Image cl.	DU-DPD	Object det.
Rate	[1..5]	[0..24]	$\frac{N}{2}$ or N	[0..9]
Figure	Fig. 4	Fig. 5b)	Fig. 9	Fig. 10
VR-PRUNE	8, 11	8, 11	5, 10	53, 72
PRUNE	17, 46	77, 100	6, 13	61, 119
SADF	60, 140	1000, 1250	9, 18	565, 920
VRDF/DPN	7, 8	7, 6	4, 7	53, 70

is so general that possibilities for graph consistency verification at design time are very limited, whereas for VRDF no practical design frameworks have been released so that the MoC's applicability to high processing performance could be evaluated. A detailed explanation of the graph size calculations is presented in Appendix B.

VII. DISCUSSION AND FUTURE WORK

Section I presented a weakly consistent dataflow graph that was originally introduced in [1], with a note that for example the TensorFlow dataflow environment produces version-dependent behavior upon execution of this graph.

How does VR-PRUNE handle this graph of Fig. 1? As such, the graph is not a valid VR-PRUNE graph. Since the graph contains dynamic token rate ports, an sDPG structure needs to be identified for VR-PRUNE compliance. Evidently, *conf* would serve as the configuration actor q, whereas *switch* and *proc* would represent the pair of dynamic actors δ associated with q. However, VR-PRUNE requires that both dynamic actors of δ need to be controlled by the same configuration actor, and hence the output of *conf* is required to be connected to *proc* as well. Now, the output port p_{sT} of *switch* and its counterpart p_{p1} in *proc* could be interpreted as DRPs controlled by the output of *conf*. With these changes, the graph would be a valid VR-PRUNE graph, and would also avoid unbounded use of memory: upon emitting a *False* token, the *conf* actor would set the atrs of the two DRPs (p_{sT} and p_{p1}) to zero, allowing *proc* to consume the token arriving from *input* independent of sample values. This example highlights the importance of formal design rules and computation models for detecting and diagnosing model flaws as early as possible.

Since one of the major attributes of VPRF is high processing performance on multicore and heterogeneous (CPU+GPU) platforms, one might wonder how VPRF compares to other similar frameworks in terms of performance. Table IV and Fig. 6 illustrate the processing performance of VPRF versus PRUNE, showing that performance differences between the frameworks are minimal, which is expectable because the differences in the runtime frameworks are modest. On the other hand, the PRUNE paper [28] presented extensive benchmarks, where it was shown that PRUNE outperformed the DAL [35] framework in all application benchmarks.

Currently, VPRF only supports a single OpenCL device in the system. As future work, this could be extended to enable multiple OpenCL devices including multiple GPUs and/or OpenCL-compatible CPUs. A further interesting direction worth exploring would be introducing distributed computing [35] for executing parts of VR-PRUNE graphs in a cloud similar to [43].

VIII. CONCLUSION

In this paper we have presented VR-PRUNE, a Model of Computation for high-performance signal processing applications, which features variable token rates and is accompanied with VPRF, a runtime library that has deeply integrated support for heterogeneous computing. VPRF is going to be released as open source similar to its predecessor PRUNE.

We have formally defined the VR-PRUNE Model of Computation, design rules, and consistency analysis, and have discussed its decidability. Compared to previous related Models of Computation, VR-PRUNE offers a unique combination of analyzability, expressiveness and practical applicability for high-performance applications.

Through extensive experiments using VPRF with four application examples, we have shown how VR-PRUNE

- Is applicable to practical signal processing algorithms,
- Offers considerably higher expressiveness than previous work,
- Enables adaptive processing for saving power, and
- Provides high processing performance.

APPENDIX A
GPU-MAPPED FIFOS

Heterogeneous computing across CPU and GPU resources needs to be implemented carefully to avoid unnecessary computation time overheads. One significant source of overhead in GPU based computation acceleration are memory transfers between the CPU and the GPU.

In VPRF, the memory structures related to dataflow actors and FIFOs reside by default in CPU memory. However, especially in machine learning applications, it is common that the application consists of a pipeline of actors (neural network layers) that are processed on the GPU. In such cases, highest performance is achieved when FIFOs between GPU-mapped actors reside in the GPU memory, avoiding unnecessary data transfers between the CPU and GPU: tokens flow within GPU memory from one GPU-mapped actor to the next GPU-mapped actor.

Although it is common practice in GPU computing to maintain intermediate data between GPU kernels ($\approx$ actors) in GPU memory buffers ($\approx$ FIFOs), implementing GPU buffering in the dataflow computing context requires some additional consideration to maintain data-driven application behavior.

In VPRF, this is achieved such that each FIFO buffer, which is mapped to GPU memory, has a dummy counterpart in CPU memory. This dummy counterpart of a FIFO does not carry any data (as the token data is in GPU memory) — it only serves for implementing synchronization between actors. By using these dummy *synchronization FIFOs*, the VR-PRUNE application simultaneously maintains data-driven behavior, while avoiding computation time overhead by keeping token data in GPU memory.

In the Image classification application, the use of GPU-mapped FIFOs decreases average image classification time from

18.7 ms to 16.2 ms when the ARMCL library is used for actor implementations on the N2 platform.

APPENDIX B
GRAPH COMPLEXITY COMPARISON

Details of the graph complexity comparison in terms of edge counts and vertex counts are explained below for PRUNE, SADF [19] and VRDF [29] models.

In terms of graph complexity, the PRUNE [28] model differs from VR-PRUNE in two significant ways: 1) PRUNE does not inherently support variable token rates, and hence token rate changes need to be emulated by a series of actor (vertex) instances that can individually be enabled or disabled, which increases graph component count compared to VR-PRUNE. 2) PRUNE does not require edges from configuration actors to those actors that are enabled/disabled at runtime. This decreases graph component count compared to VR-PRUNE. For an illustration related to these differences, the reader should refer to Fig. 4 whose subfigures a) and b) show equivalent PRUNE and VR-PRUNE graphs. Related to the PRUNE model of the Fig. 5 parallel image classifier, it is important to notice that subfigures a) and b) do *not* depict graphs of equivalent behavior. A PRUNE graph equivalent to the Fig. 5(b) VR-PRUNE graph would consist of 24 instances of actors a, b, and c, which would result in the 77 actors and 100 edges, as shown in Table VI.

The SADF model captures dynamic (runtime) changes on graph topology by scenarios such that each possible topology is described by a separate SDF graph. In the Adaptive DPD application the number of active filter branches ranges between 1 and 5, resulting in 5 scenarios (graphs), each of which has a different number of actors and edges in the branches: $S^{adpd}_{edges} = \Sigma^{N}_{i=1} 8i + 4$, and $S^{adpd}_{vertices} = \Sigma^{N}_{i=1} 2i + 6$. For $N = 5$, a total of 60 actors and 140 edges ensue when all scenarios are added together. In the Image classification application, between 0 and $N = 24$ images can be classified in parallel, resulting in 25 graph scenarios: $S^{imcl}_{edges} = \Sigma^{N}_{i=0} 4i + 2$, and $S^{imcl}_{vertices} = \Sigma^{N}_{i=0} 3i + 4$, which totals into 1000 actors and 1250 edges (the numbers are exact) for $N = 24 + 1$. For DU-DPD the number of scenarios is $N = 2$ ($S^{dudpd}_{edges} = \Sigma^{N}_{i=1} 4i + 3$, and $S^{dudpd}_{vertices} = \Sigma^{N}_{i=1} i + 3$), and for Object detection there are $N + 1 = 10$ scenarios: $S^{objd}_{edges} = \Sigma^{N}_{i=0} 6i + 65$, and $S^{objd}_{vertices} = \Sigma^{N}_{i=0} i + 52$. Since SADF does not explicitly mention the need of configuration actors, the component counts do not include the configuration actor, or edges connected to the configuration actor.

Finally, the VRDF [29] and DPN [21] models are slightly more expressive than VR-PRUNE: neither of these models requires symmetric token rates, nor any need for configuration actors. To this extent both VRDF and DPN are able to directly capture VR-PRUNE graphs, albeit without configuration actors and edges. Consequently VRDF and DPN graph complexities are very similar, but slightly lower than those of VR-PRUNE graphs.

REFERENCES

[1] J. T. Buck, "Scheduling dynamic dataflow graphs with bounded memory using the token flow model," Ph.D. dissertation, Dept. Elect. Eng. and Comput. Eng., Univ. California Berkeley, 1993.

[2] G. F. Zaki, W. Plishker, S. S. Bhattacharyya, C. Clancy, and J. Kuykendall, "Integration of dataflow-based heterogeneous multiprocessor scheduling techniques in GNU radio," *J. Signal Process. Syst.*, vol. 70, no. 2, pp. 177–191, 2013.

[3] M. Abadi *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," 2016, *arXiv:1603.04467*.

[4] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Trans. Comput.*, vol. C-36, no. 1, pp. 24–35, Jan. 1987.

[5] K. Ravindran, A. Ghosal, R. Limaye, G. Wang, G. Yang, and H. Andrade, "Analysis techniques for static dataflow models with access patterns," in *Proc. Conf. Des. Architectures Signal Image Process.*, 2012, pp. 1–8.

[6] *Agilent EEsof EDA SystemVue 2011 Technical Overview*, Agilent Technologies, Inc., May 2011.

[7] A. Veit and S. Belongie, "Convolutional networks with adaptive inference graphs," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 3–18.

[8] R. T. Mullapudi, W. R. Mark, N. Shazeer, and K. Fatahalian, "Hydranets: Specialized dynamic architectures for efficient inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8080–8089.

[9] J. T. Buck and E. A. Lee, "Scheduling dynamic dataflow graphs with bounded memory using the token flow model," in *Proc. Int. Conf. Acoust., Speech, Signal Process.*, 1993, vol. 1, pp. 429–432.

[10] Y. Yu *et al.*, "Dynamic control flow in large-scale machine learning," in *Proc. 13th EuroSys Conf.*, 2018, pp. 1–15.

[11] Y. Ma, J. Wu, S. S. Bhattacharyya, and J. Boutellier, "Decidable variable-rate dataflow for heterogeneous signal processing systems," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2020, pp. 1683–1687.

[12] J. L. Peterson, "Petri nets," *ACM Comput. Surv.*, vol. 9, no. 3, pp. 223–252, 1977.

[13] K. M. Kavi, B. P. Buckles, and U. N. Bhat, "A formal definition of data flow graph models," *IEEE Trans. Comput.*, vol. 35, no. 11, pp. 940–948, Nov. 1986.

[14] K. M. Kavi, B. P. Buckles, and U. N. Bhat, "Isomorphisms between Petri nets and dataflow graphs," *IEEE Trans. Softw. Eng.*, vol. SE-13, no. 10, pp. 1127–1134, Oct. 1987.

[15] M. Pelcat, K. Desnos, J. Heulot, C. Guy, J.-F. Nezan, and S. Aridhi, "PREESM: A dataflow-based rapid prototyping framework for simplifying multicore DSP programming," in *Proc. Eur. Embedded Des. Educ. Res. Conf.*, 2014, pp. 36–40.

[16] W. Thies, M. Karczmarek, and S. Amarasinghe, "StreamIt: A language for streaming applications," in *Proc. Int. Conf. Compiler Construction*, Springer, 2002, pp. 179–196.

[17] H. P. Huynh, A. Hagiescu, O. Z. Liang, W.-F. Wong, and R. S. M. Goh, "Mapping streaming applications onto GPU systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 9, pp. 2374–2385, Sep. 2014.

[18] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete, "Cyclo-static dataflow," *IEEE Trans. Signal Process.*, vol. 44, no. 2, pp. 397–408, Feb. 1996.

[19] S. Stuijk, M. Geilen, B. Theelen, and T. Basten, "Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications," in *Proc. Int. Conf. Embedded Comput. Syst.*, 2011, pp. 404–411.

[20] E. Jeong, D. Jeong, and S. Ha, "Dataflow model-based software synthesis framework for parallel and distributed embedded systems," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 26, no. 5, pp. 1–38, 2021.

[21] E. A. Lee and T. M. Parks, "Dataflow process networks," *Proc. IEEE*, vol. 83, no. 5, pp. 773–801, May 1995.

[22] J. Eker and J. W. Janneck, "CAL language report," UC Berkeley, Tech. Rep. UCB/ERL M03/48, 2003.

[23] M. Mattavelli, I. Amer, and M. Raulet, "The reconfigurable video coding standard [standards in a nutshell]," *IEEE Signal Process. Mag.*, vol. 27, no. 3, pp. 159–167, May 2010.

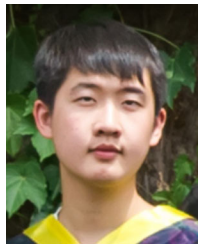
[24] G. Cedersjö and J. W. Janneck, "Tÿcho: A framework for compiling stream programs," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 6, pp. 1–25, 2019.

[25] H. Yviquel, A. Lorence, K. Jerbi, G. Cocherel, A. Sanchez, and M. Raulet, "ORCC: Multimedia development made easy," in *Proc. ACM Int. Conf. Multimedia*, 2013, pp. 863–866.

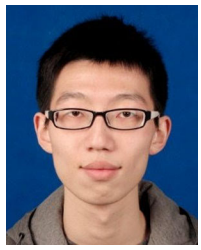
[26] O. Rafique and K. Schneider, "SHeD: A framework for automatic software synthesis of heterogeneous dataflow process networks," in *Proc. Euromicro Conf. Digit. Syst. Des.*, 2020, pp. 1–10.

[27] G. Gao, R. Govindarajan, and P. Panangaden, "Well-behaved dataflow programs for DSP computation," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 1992, vol. 5, pp. 561–564.
[28] J. Boutellier, J. Wu, H. Huttunen, and S. S. Bhattacharyya, "PRUNE: Dynamic and decidable dataflow for signal processing on heterogeneous platforms," *IEEE Trans. Signal Process.*, vol. 66, no. 3, pp. 654–665, Feb. 2018.
[29] M. H. Wiggers, M. J. Bekooij, and G. J. Smit, "Buffer capacity computation for throughput constrained streaming applications with data-dependent inter-task communication," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp.*, 2008, pp. 183–194.
[30] L. Schor, A. Tretter, T. Scherer, and L. Thiele, "Exploiting the parallelism of heterogeneous systems using dataflow graphs on top of OpenCL," in *Proc. IEEE Symp. Embedded Syst. Real-time Multimedia*, 2013, pp. 41–50.
[31] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proc. IEEE*, vol. 75, no. 9, pp. 1235–1245, Sep. 1987.
[32] J. L. Pino, S. S. Bhattacharyya, and E. A. Lee, "A hierarchical multiprocessor scheduling system for DSP applications," in *Proc. Asilomar Conf. Signals, Syst. Comput.*, 1996, vol. 1, pp. 122–126.
[33] J. McAllister and M. Davis, "Graph coordination for compact representation of regular dataflow structures," in *Proc. IEEE Workshop Signal Process. Syst.*, 2020, pp. 1–6.
[34] E. A. Lee and S. Ha, "Scheduling strategies for multiprocessor real-time DSP," in *Proc. IEEE Glob. Telecommun. Conf. Exhibition 'Commun. Technol. 1990 s Beyond'*, 1989, pp. 1279–1283.
[35] L. Schor, I. Bacivarov, D. Rai, H. Yang, S.-H. Kang, and L. Thiele, "Scenario-based design flow for mapping streaming applications onto on-chip many-core systems," in *Proc. Int. Conf. Compilers, Architectures Synth. Embedded Syst.*, 2012, pp. 71–80.
[36] B. Gaster, L. Howes, D. R. Kaeli, P. Mistry, and D. Schaa, *Heterogeneous Computing With openCL: Revised openCL 1 1.2 edition* Newnes, 2012.
[37] P. P. Campo, V. Lampu, A. Meirhaeghe, J. Boutellier, L. Anttila, and M. Valkama, "Digital predistortion for 5G small cell: GPU implementation and RF measurements," *J. Signal Process. Syst.*, vol. 92, no. 5, pp. 475–486, 2020.
[38] M. Aghababaeetafreshi, D. Korpi, M. Koskela, P. Jääskeläinen, M. Valkama, and J. Takala, "Software defined radio implementation of a digital self-interference cancellation method for inband full-duplex radio using mobile processors," *J. Signal Process. Syst.*, vol. 90, no. 10, pp. 1297–1309, 2018.
[39] L. Anttila, P. Handel, and M. Valkama, "Joint mitigation of power amplifier and I/Q modulator impairments in broadband direct-conversion transmitters," *IEEE Trans. Microw. Theory Techn.*, vol. 58, no. 4, pp. 730–739, Apr. 2010.
[40] K. Muhammad, S. Khan, V. Palade, I. Mehmood, and V. H. C. De Albuquerque, "Edge intelligence-assisted smoke detection in foggy surveillance environments," *IEEE Trans. Ind. Informat.*, vol. 16, no. 2, pp. 1067–1075, Feb. 2020.
[41] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.
[42] W. Liu *et al.*, "SSD: Single shot multibox detector," in *Proc. Eur. Conf. Comput. Vis.*, Cham, Switzerland: Springer, 2016, pp. 21–37.
[43] Y. Kang *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Comput. Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

Jani Boutellier (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees from the University of Oulu, Oulu, Finland, in 2005 and 2009, respectively. He is currently an Associate Professor with the School of Technology and Innovations, University of Vaasa, Vaasa, Finland. In 2007-2008, 2013 he was a Visiting Researcher with the Swiss Federal Institute of Technology Lausanne, Lausanne, Switzerland. His research interests include dataflow programming, signal processing, heterogeneous computing and machine learning for computer vision. Between 2016 and 2021, he was a Member of the IEEE Signal Processing Society Design and Implementation of Signal Processing Systems (DISPS/ASPS) Technical Committee.



Yujunrong Ma received the bachelor's degree in automation from the Harbin Institute of Technology, Harbin, China, the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Maryland, University of Maryland, College Park, MD, USA. He held third position with the National Institute of Justice (NIJ) recidivism prediction challenge. His research interests include dataflow implementations, deep learning and evolutionary algorithms. He was the recipient of the Dean's Fellowship in 2019.



Jiahao Wu received the bachelor's degree from the University of Electronic Science and Technology of China, Chengdu, China, the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Maryland, College Park, MD, USA. His research interests include model-based design for parallel computing, dataflow implementations and synthesis of digital signal processing systems.

Mir Khan received the bachelor's degree from the University College of Bahrain, Janabiyah, Bahrain, the master's degree from Tampere University, Tampere, Finland, where he is currently working toward the Ph.D. degree. His research interests include optimizing neural networks inference implementations for graphical processing units and embedded systems.

Shuvra S. Bhattacharyya (Fellow, IEEE) received the Ph.D. degree from the University of California, Berkeley, Berkeley, CA, USA. He is currently a Professor with the Department of Electrical and Computer Engineering, University of Maryland, College Park, MD, USA. He holds a joint appointment with the University of Maryland Institute for Advanced Computer Studies, and is affiliated with the Maryland Crime Research and Innovation Center. He also holds a part time visiting position as the Chair of Excellence in Design Methodologies and Tools with the Institut National Des Sciences Appliquées, Rennes, France. He has held industrial positions as a Researcher with the Hitachi America Semiconductor Research Laboratory, and Compiler Developer at Kuck & Associates. From 2015 to 2018, he was a part time Visiting Professor with the Department of Pervasive Computing, Tampere University of Technology (now Tampere University), Tampere, Finland, as part of the Finland Distinguished Professor Programme. He has also held Visiting Research positions with the U.S. Air Force Research Laboratory.