Decidable Variable-Rate Dataflow for Heterogeneous Signal Processing Systems

DECIDABLE VARIABLE-RATE DATAFLOW FOR HETEROGENEOUS SIGNAL PROCESSING SYSTEMS

Yujunrong Ma, Jiahao Wu,
Shuvra S. Bhattacharyya

Dept. of Electrical & Computer Engineering,
Institute for Advanced Computer Studies
University of Maryland, College Park, USA

Jani Boutellier

School of Technology and Innovations,
University of Vaasa
Vaasa, Finland

ABSTRACT

Dynamic dataflow models of computation have become widely used through their adoption to popular programming frameworks such as TensorFlow and GNU Radio. Although dynamic dataflow models offer more programming freedom, they lack analyzability compared to their static counterparts (such as synchronous dataflow).

In this paper we advocate the use of a boundedly dynamic dataflow model of computation, VR-PRUNE, that remains analyzable but still offers more programming freedom than a fully static dataflow model.

The paper presents the VR-PRUNE model of computation and runtime, and illustrates its applicability to practical signal processing applications by two use cases: an adaptive convolutional neural network, and a predistortion filter for wireless communications. By runtime experiments on two heterogeneous computing platforms we show that VR-PRUNE is both flexible and efficient.

***Index Terms*—** variable-rate dataflow, models of computation, signal processing, heterogeneous computing

1. INTRODUCTION

In the recent years, dataflow has been used as the underlying model of computation (MoC) in TensorFlow for machine learning applications [1], and in GNU Radio for radio communications [2]. Dataflow MoCs provide an abstract and efficient for way describing various signal processing applications. Under dataflow, an application is described as a graph that consists of nodes and directed edges that interconnect the nodes. The nodes, called *actors*, perform computations, whereas the edges act as order-preserving communication channels between actors. The data that is communicated over the edges is packaged into *tokens* of fixed size. An actor can trigger (*fire*) its computations when its input edges have a sufficient number of tokens available – this token quantity is called the *input token rate*. Respectively, during its firing the actor produces a number of tokens to each of its output channels. The channel-specific number of tokens produced is called *output token rate*.

In one of the most studied dataflow MoCs, Synchronous Dataflow (SDF) [3], the data quantities produced and consumed by each actor need to be fixed and known at application design time. This restriction however enables a variety of compile-time optimizations, as well as full consistency checks at design time for, e.g., ensuring that the application cannot deadlock [4] or cause communication buffer overflows. If the dataflow MoC allows such a full consistency check in finite time, the MoC is called *decidable*. Consequently, SDF has been used for specifying a variety of complex digital signal processing systems [5] [6][7].

Despite the advantages of fully static dataflow MoCs, there are situations where fixed token rates are too restrictive. Examples of such a case are adaptive signal processing applications where the application behavior changes at runtime in response to changes in the environment. An example of such an application, used in this paper, is a neural network for video analysis that can adjust its inference frequency based on video motion detection.

In order to properly support such dynamic behavior, a dynamic dataflow MoC is needed. Consequently, actors' token rates can vary at runtime in response to dynamically varying computation needs. As a drawback, such flexibility restricts application analyzability. An example of a dynamic dataflow model is Boolean dataflow [8] that is not a decidable MoC.

Between the extremes of fully static and fully dynamic dataflow, a variety of boundedly dynamic dataflow MoCs exist, of which a well-known example is Well-Behaved Dataflow (WBDF) [9]. Recently, Boutellier et al. presented the PRUNE MoC and framework [10], which similar to WBDF allows dynamic application behavior to be expressed within dataflow subgraphs that follow a constrained graph topology. A different branch of work is variable-rate dataflow (VRDF) [11] that allows port token rates to change arbitrarily within predefined limits.

In this paper, we introduce a novel dataflow MoC that is

This work has been partially funded by Academy of Finland / 326383.

a hybrid between the PRUNE MoC [10] and the VRDF MoC [11]: VR-PRUNE keeps the decidability of PRUNE, but adds to it support for variable token rates. *Here we show that VR-PRUNE increases MoC flexibility over the PRUNE MoC without having a negative impact on run-time efficiency*. Some early results related to the proposed MoC were presented in our previous work [12], while in this paper the VR-PRUNE MoC is formalized (as far as the available space allows), and the MoC's run-time efficiency is illustrated by experiments.

2. BACKGROUND: THE PRUNE MOC

In this section, we briefly present the PRUNE MoC [10] that acts as the basis of the proposed VR-PRUNE MoC.

In PRUNE, an application is expressed as a graph $G = (A, F)$, where A is a set of actors and F is a set of FIFO channels. A FIFO channel is connected to an actor through an actor *port*; each actor $a \in A$ can have any number of input and output ports. We use the following notation: $a = parent(p)$ denotes an actor a contains port p, and p^+_{a1} stands for output port 1 of actor a. The superscript $+/-$ corresponds to output/input port direction, respectively.

For FIFO $f \in F$, we say ports p^+_a and p^-_b are *connected* if $fifo(p^+_a) = fifo(p^-_b)$, where actors a and b are referred as the source and sink of the same FIFO. In the PRUNE MoC, an output port p^+ can be connected to multiple FIFOs, but every input port p^- has only one FIFO connected to it and each FIFO has unique source and sink ports.

There are three types of ports in the PRUNE MoC: the control (input/output) port, the static regular port (SRP) and the dynamic regular port (DRP). SRPs have a fixed token consumption/production rate, whereas DRPs have two fixed token rates, which are respectively denoted as *active token rate* and *inactive token rate* of p, also abbreviated as $atr(p)$ and $itr(p)$. Morever, each FIFO has a single, positive-integer token rate, denoted as $fiforate(f)$, and we restrict that $atr(p) = fiforate(fifo(p))$, in other words the active port rate must strictly equal to the token rate of the connected FIFO. By this *symmetric-rate dataflow* behavior, PRUNE requires the token consumption rate equal the production rate for every FIFO. Even though this makes PRUNE more restricted than SDF with respect to token rate conversion, PRUNE on the other hand allows dynamic token rates that are not supported by SDF.

Two PRUNE actors are $adjacent$ if each of them has at least one port connected over the same FIFO. A *chain* $S = (a_1, a_2, ..., a_n)$ is a set of actors for which $\forall i = 1, 2, ..., n$, a_i and a_{i+1} are adjacent, and S connects a_1 and a_n.

3. THE PROPOSED VR-PRUNE MOC

The proposed VR-PRUNE MoC is a generalization of the PRUNE MoC in that VR-PRUNE allows token rates of certain ports to dynamically vary within predefined limits, similar to VRDF [11]. In the following subsections, we identify the new structures and required changes compared to the PRUNE MoC to enable this variable-token rate behavior.

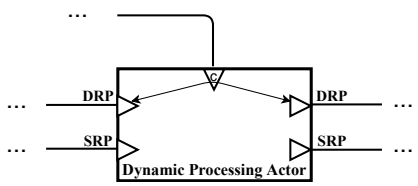

Fig. 1. An illustrative example of a dynamic processing actor.

3.1. VR-PRUNE actor types

The varying token rate feature requires a new type of actor, the *dynamic processing actor*, that is introduced among other actor types that are identical to those in PRUNE.

a. Static Processing Actor: Static processing actors (SPA) can only have SRP ports, and therefore during every firing of an SPA all ports p must have the token rate $atr(p)$.

b. Configuration Actor: A configuration actor can have one or multiple control output ports which must be connected to the control input port of a dynamic actor or a dynamic processing actor. The control output ports must be SRPs with a token rate of unity. Besides, a configuration actor may have zero or more input or output data ports of type SRP.

c. Dynamic Actor: A dynamic actor has at least one DRP, one control input port and any number of SRPs. All the DRPs have to be either input ports, or output ports. Every time a dynamic actor performs a firing, a token is consumed from each of its control input ports. *In contrast to PRUNE, values of tokens originating from the control input port set the token rate of each DRP p to any integer value between 0 and* $atr(p)$.

d. Dynamic Processing Actors (DPA): DPAs are similar to dynamic actors, but differ in the sense that a DPA is required to have at least one input DRP and at least one output DRP, as well as at least one control input port. Also, a DPA can have any number of SRPs. Similar to dynamic actors, the control input port sets the token rates of DRPs to any integer value between 0 and $atr(p)$.

Fig. 1 provides an illustration of a DPA: the control input port sets the token rates of a pair of DRPs (one on the input side, the other on the output side). When the actor performs a firing, the value of a single token coming from the control port sets the token rate of both DRPs.

3.2. VR-PRUNE design rules

Similar to PRUNE, VR-PRUNE imposes a small set of concrete design rules to ensure that graph topologies leading to inconsistent dataflow behavior are avoided. This is achieved by five *design rules* that are identical to those in PRUNE, except for the *connecting subchain* rule (3).

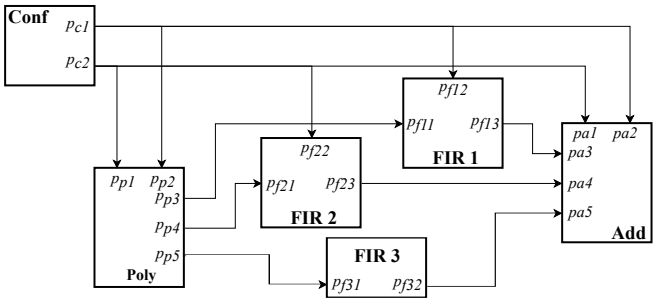

Fig. 2. A VR-PRUNE dynamic processing graph.

1. Linked port control rule: A linked pair of DRPs $\{p_x, p_y\}$ must be controlled by the same control output port.

2. Balanced delay rule: If a control output port p_c controls two control input ports p_x, p_y, then $delay(p_c, p_x) = delay(p_c, p_y)$.

3. Connecting subchain rule: If $\{p_x, p_y\}$ are DRPs linked by the chain $S = (a_1, a_2, ..., a_n)$, then (1) $a_i, i = 1, 2, ..., n$ must all be actors of type SPA or DPA; (2) every connecting subchain to which $a_i \in S$ belongs, must be associated with the two dynamic actors x and y.

4. Single-sided dynamism rule: A dynamic actor can only have either input DRPs or output DRPs, not both.

5. Encapsulation rule: If $\{p_x, p_y\}$ are linked DRPs whose parents are x and y, $S = (a_1, a_2, ..., a_n)$ is a chain, and $k \notin S$ is an actor that connects to $a_i, i = 1, 2, ..., n$, then k must belong to a chain that connects x and y.

3.3. VR-PRUNE subgraphs and control tables

The VR-PRUNE design rules require that dynamic token rates occur within a specific type of subgraph, the Dynamic Processing Graph (DPG). A DPG consists of one configure actor, two dynamic actors and any number of DPAs or SPAs. Below, an example of a DPG is described.

Fig. 2 shows an example of a VR-PRUNE DPG, where FIR1 and FIR2 are dynamic processing actors, p_{f11}, p_{f13} and p_{f21}, p_{f23} are DRPs, controlled by p_{f12}, p_{f22} respectively. The Poly actor is a dynamic actor with DRPs on the output side and the Add actor is also a dynamic actor with DRPs on the input side. Conf is a configuration actor and FIR3 is a static processing actor.

The values of tokens produced by the configuration actor's control output ports can be illustrated by a *control table*. Table 1 shows an example of a VR-PRUNE control table, which is a $h \times w$ matrix, where h and w are the numbers of control output ports and DRPs. All actors between a linked pair of DRPs $\{p_x, p_y\}$ are required to be controlled by the same control output port of the configuration actor. In Fig 2, p_{c1} is connected to p_{p2}, p_{f12} and p_{a2}, which actually controls DRPs $p_{p3},\ p_{f11},\ p_{f13}$ and p_{a3}, respectively. This control relationship can be seen in the Table 1 control table such that for the rows of p_{c1} only the DRPs controlled by p_{c1} have values, which reflect the range of allowed token rates.

Table 1. Control table for the graph in Fig. 2.

	p_{p3}	p_{p4}	p_{f11}	p_{f13}
p_{c1}	[0 .. 2]	-	[0 .. 2]	[0 .. 2]
p_{c2}	-	[0 .. 3]	-	-
	p_{f21}	p_{f23}	p_{a3}	p_{a4}
p_{c1}	-	-	[0 .. 2]	-
p_{c2}	[0 .. 3]	[0 .. 3]	-	[0 .. 3]

3.4. VR-PRUNE MoC summary

The novelty of VR-PRUNE over PRUNE is the support for token rates that vary within predefined limits. Whereas in PRUNE DRP ports had two token rates, *itr* and *atr*, VR-PRUNE allows these ports to have any integer token rate between *itr* and *atr*. To realize this, VR-PRUNE introduces a new actor type, the DPA, as well as a modification to the *connecting subchain* design rule. Unfortunately, due to strict space limits, it is not possible to show decidability proofs of the VR-PRUNE MoC here.

In the next section, we will show by experiments with the VR-PRUNE runtime that this added design and MoC freedom of VR-PRUNE does not add computational overhead in the context of heterogeneous target platforms.

4. EXPERIMENTAL RESULTS

In this section, we present two application use cases for VR-PRUNE and measure the efficiency of the VR-PRUNE runtime. Similar to PRUNE, VR-PRUNE is designed for heterogeneous systems that use both a GPU and multiple CPU cores for computations. The considered target platforms are shown in Table 2. The designer provides actor descriptions in C or OpenCL, based on which the VR-PRUNE compiler generates a toplevel file that realizes inter-actor communication.

4.1. Dynamic-update predistortion filter

The dynamic-update predistortion filter (DU-DPD) is a filter for wireless transmitters that in real-time filters the baseband signal to be transmitted. The VR-PRUNE implementation of the DU-DPD has an online learning capability, which allows updating the filter coefficients periodically to accommodate with varying RF interference.

As Fig. 3 shows, actors Src, $Learn$ and $Update$ form the varying-token rate learning chain, where Src and $Update$ are dynamic actors, and $Learn$ is a DPA. This dynamic-token rate feature allows any number of samples between 1 and X to be used as training data to acquire new filter coefficient values. The actor $Filter$ on the other hand operates in real time on the GPU at a constant token rate of N

Table 2. Platforms used for experiments.

Tag	GPPs	GPU	Operating System
i7-940MX	Intel i7-6700HQ @ 2.60 GHz	NVidia GeForce 940MX	Ubuntu 18.04, g++ 7.0.0
i7-GTX1080	Intel i7-8700K @ 3.70GHz	NVidia Geforce GTX1080	Ubuntu 18.04, g++ 7.0.0

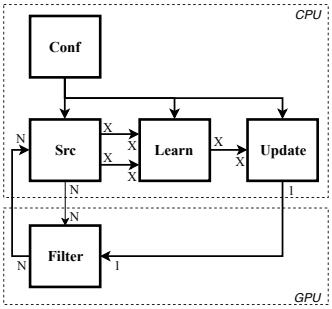

Fig. 3. VR-PRUNE model for the DU-DPD use case.

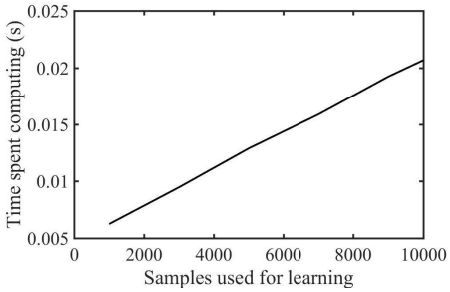

Fig. 4. Time spent in DU-DPD under VR-PRUNE on i7-GTX1080, as a function of samples used for learning.

Fig. 4 shows the execution time of the whole DU-DPD graph for one iteration as a function of *kilo*samples used (X) for training in the $Learn$ actor. Here, the minimum value of X was 1, and the maximum value was 10; intermediate sample rates were varied with a stepping of 1. The figure shows that in this applications the variable sample rate feature allows reducing the number of samples used for learning in situations when there is little or no RF interference, with significant computation time savings.

4.2. Adaptive Convolutional Neural Network

The dataflow graph of our neural network use case is depicted in Fig. 5; it consists of two GPU-accelerated convolution layers (L1Conv, L2Conv) followed by a GPU-accelerated dense layer, as well as two more dense layers that have been combined into a single actor (L3Relu-L5). The application is adaptive in the sense that it allows dynamically enabling and disabling convolutional neural network inference for individual frames, e.g. based on inter-frame motion detection.

In the dataflow sense, the image data related to one frame

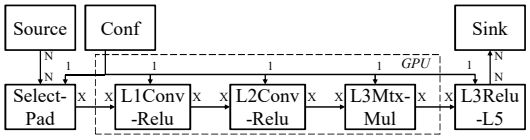

Fig. 5. The adaptive CNN application.

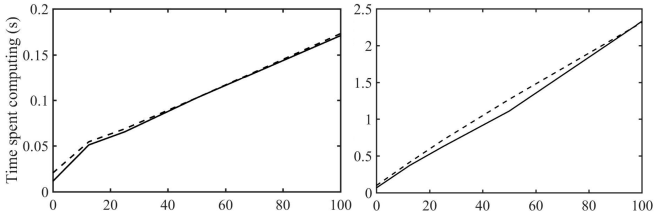

Fig. 6. Time spent in the CNN application as a function of % of frames processed: PRUNE (dashed line) vs. VR-PRUNE (solid line) on i7-GTX1080 (left) and i7-940MX (right).

(image) forms a single token. For accelerating GPU processing, $N = 24$ frames are processed in parallel, but for each of the 24 frames, the inference can be disabled, which makes the token rate vary in steps of 1 between values 0 and 24. Fig. 5 shows the varying-token rates of ports by X (= [0, 24]) and fixed rates by N (= 24).

Fig. 6 shows the processing time of the whole graph for a sequence of 384 frames on two platforms (Table 2). The CNN inference (on/off) for each frame was randomly varied at runtime achieving the average percentage of frames processed for 0%, 12.5%, 25.0%, 50.0% and 100%, measuring the execution time time for each percentage value.

For this use case the conditional CNN inference feature was implemented under conventional PRUNE using the *conditional statements* approach (See [12]) and by the variable token rate feature of VR-PRUNE. The results show that the variable token rate feature of VR-PRUNE does not impose any overhead compared to conventional PRUNE – in contrast, VR-PRUNE is computationally slightly more efficient than conventional PRUNE due to the fact that variable data rate processing allows omitting data transfer for skipped frames.

5. CONCLUSION

We have introduced VR-PRUNE, a Model of Computation and runtime for signal processing on heterogeneous platforms. Experimental results show that VR-PRUNE is both more flexible and efficient than its predecessor PRUNE.

6. REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[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," *Journal of Signal Processing Systems*, vol. 70, no. 2, pp. 177–191, 2013.

[3] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Transactions on Computers*, vol. 100, no. 1, pp. 24–35, 1987.

[4] E. A. Lee and S. Neuendorffer, "Concurrent models of computation for embedded software," *IEE Proceedings-Computers and Digital Techniques*, vol. 152, no. 2, pp. 239–250, 2005.

[5] S. Ritz, M. Pankert, V. Zivojinovic, and H. Meyr, "Optimum vectorization of scalable synchronous dataflow graphs," in *Proceedings of International Conference on Application Specific Array Processors*. IEEE, 1993, pp. 285–296.

[6] B. Bhattacharya and S. S. Bhattacharyya, "Quasi-static scheduling of reconfigurable dataflow graphs for DSP systems," in *Proceedings of International Workshop on Rapid System Prototyping*. IEEE, 2000, pp. 84–89.

[7] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, *Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems*. USA: Kluwer Academic Publishers, 2001, pp. 527–543.

[8] J. T. Buck and E. A. Lee, "Scheduling dynamic dataflow graphs with bounded memory using the token flow model," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 1. IEEE, 1993, pp. 429–432.

[9] G. Gao, R. Govindarajan, and P. Panangaden, "Well-behaved dataflow programs for DSP computation," in *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 5. IEEE, 1992, pp. 561–564.

[10] J. Boutellier, J. Wu, H. Huttunen, and S. S. Bhattacharyya, "PRUNE: Dynamic and decidable dataflow for signal processing on heterogeneous platforms," *IEEE Transactions on Signal Processing*, vol. 66, no. 3, pp. 654–665, 2017.

[11] 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 *IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE, 2008, pp. 183–194.

[12] J. Boutellier and S. S. Bhattacharyya, "Low-power heterogeneous computing via adaptive execution of dataflow actors," in *IEEE International Workshop on Signal Processing Systems (SiPS)*. IEEE, 2017, pp. 1–6.