A METHODOLOGY FOR OPTIMIZING BUFFER SIZES OF DYNAMIC DATAFLOW FPGAS IMPLEMENTATIONS

Ab Al-Hadi Ab Rahman, Simone Casale-Brunet , Claudio Alberti, Marco Mattavelli*

EPFL SCI STI MM
École Polytechnique Fédérale de Lausanne, Switzerland

ABSTRACT

Minimizing buffer sizes of dynamic dataflow implementations without introducing deadlocks or reducing the design performance is in general an important and useful design objective. Indeed, buffer sizes that are too small causing a system to deadlock during execution, or dimensioning unnecessarily large sizes leading to a resource inefficient design are both not a desired design option. This paper presents an implementation, validation, and comparison of several buffer size optimization techniques for the generic class of dynamic dataflow model of computation called the dataflow process network. The paper presents an heuristic capable of finding a close-to-minimum buffer size configuration for deadlock-free executions, and a methodology to efficiently explore different configurations for feasible design alternatives. The approach is demonstrated using as experimental design case, an MPEG-4 AVC/H.264 decoder implemented on an FPGA.

Index Terms— Buffer size optimization, CAL dataflow specifications, MPEG-4 decoder, FPGA

1. INTRODUCTION

The Dataflow Process Network (DPN) [1] is a Model of Computation (MoC) that can be represented as a network of processing elements called nodes implementing the processes and edges representing the interconnections and implemented as unidirectional FIFO channels. The processing nodes in a DPN is typically referred to as *actors*, that encapsulate their own state and do not share memory with one another. Actors only communicate with each other *exclusively* by sending and receiving tokens along the FIFO channels connecting them. The DPN MoC does not impose any restriction on the actor execution, where execution scheduling in general can depend on the input data. For this reason the execution of a DPN dataflow network in general cannot not be scheduled at compile-time, and the bounds for the FIFO interconnections, information necessary for actual implementations, cannot not be determined statically, unlike other more restricted (and much less expressive) MoC such as the synchronous dataflow (SDF) and their variants [2]. In general, the problem of dimensioning buffers for a DPN network is considered unfeasible, however in most of streaming application cases such as in video and audio coding for instance, it is meaningful and useful to determine buffer size configurations satisfying a set of statistically representative input stimuli that covers with extremely high probability the typical use cases.

In this work, two optimization approaches are implemented, validated, and compared: the first is the classical approach where an analysis is performed on the dataflow program itself, and the second is based on a recently proposed approach based on post-processing the information contained in the so-called execution trace. For each of these approaches, the objectives are 1) to find a close-to-minimum buffer size configuration, and 2) to find larger optimized configurations for higher throughput design requirements. As design case, the subset of the CAL dataflow language [3] standardized by ISO/MPEG has been used to design an MPEG-4 AVC/H.264 decoder [4]. The high-level CAL dataflow program implementing the AVC decoder has been synthesized directly to HDL for obtaining an FPGA implementation, where the buffer size configuration had to be determined.

1.1. RELATION TO PRIOR WORK

Relatively few works concerning buffer size optimization techniques applied to DPN MoC can be found in literature, if we compare with the vast literature devoted to SDF such as [5], [6], [7] and related references. Most of the techniques used for both SDF and DPN, follow the same approach, i.e. by employing some determined static and dynamic scheduling strategies respectively applied to SDF and DPN networks. Another approach is based on the analysis of the execution of an application, obtained from a representative input stimuli set. This work completes by comparing some traditional approaches presented in literature with the initial results of the alternative approach presented in [8]. Moreover, this work validates the buffer size optimization techniques, by applying it to a complex implementation of an MPEG-4 AVC/H.264 decoder as design case. Finally, the work also presents comparisons with some state-of-the-art approach in buffer size optimizations.

* Sponsored by the Fonds National Suisse pour la Recherche Scientifique, grant 200021.138214.

978-1-4799-2893-4/14/$31.00 ©2014 IEEE

2. BUFFER SIZE OPTIMIZATION: CLASSICAL STATE-OF-THE-ART APPROACH

The simplest way to assign buffer sizes to the communication channels for the execution of a DPN network is based on transforming the network G to produce a semantically equivalent network G^0 that is bounded by b^0 (size for all interconnections). This transformation may introduce deadlock [9] such that the network G^0 represents only a partial execution of the original network G. However, if the execution of the network G^0 never stops and results in a complete execution, then we have succeeded in implementing a complete and bounded execution. If the execution of G^0 stops, it means that the chosen bound b^0 is too small. One or more of the channels require a buffer size larger than b^0, therefore, a larger bound $b^1 > b^0$ must be chosen. The bound can be relaxed successively, $b^0 < b^1 < b^2 < \ldots$ until a complete execution is obtained. By definition, there exists a bound b that is finite if the network G can be executed in finite time. After N iterations, a bound $b^N \leq b^0$ that corresponds to a complete execution of the network G^N is obtained. The essential problem of this approach is that all communication channels results in the same capacity value, which in most cases, is not required for a complete and deadlock-free execution. This also results in a waste of platform resources.

2.1. Finding close-to-minimum buffer configuration

In the following, the techniques proposed in [7, 9] are summarized. The initial capacities of all channels are set to 1, and a simulation is performed. If execution stops due to deadlock, it means that one or more actors are blocked writing to a full channel. Increasing the capacity of channels that are not full does not allow execution to continue. It is necessary to increase the capacity of one or more full channels. It is important not to increase the largest buffer because this could lead to unbounded growth of that buffer. Therefore, only the capacity of the *smallest* full channel is increased, because this guarantees that every full channel will eventually be increased if necessary to unlock the program. The reasoning is that if the same channel is increased repeatedly, then eventually it will no longer be the smallest full channel. If some full channel other than the smallest full channel is increased, then some buffer could grow without bound. The way to prevent this from happening is to increase the smallest full channel and to avoid additional tokens being added to destination actions with already sufficient input tokens. This technique has been implemented using a Modelsim TCL script for a fully automated approach applied at the hardware execution level.

2.2. Buffer configuration with throughput constraints

Buffer size configurations that are close-to-minimum typically do not yield a design with high streaming throughput. This is due to the fact that actor executions depend on the availability of input tokens. Small buffer sizes would likely result in a lower level of parallel actor executions due to the scarce availability of data tokens in the network.

This subsection describes an approach that finds a close-to-minimum buffer size configuration given a throughput constraint. Starting from a network with bound b for all interconnections, the buffer sizes can be limited either directly [10, 11] or indirectly [12] using feedback channels such that a lower overall buffer size can be obtained, but at the same time achieving the same throughput achievable by the configuration with a buffer size with bound b. The following describes the direct buffer size reduction technique, which is suited for hardware implementation target with implicit control.

First, a large enough buffer bound b is searched for, such that a relatively high throughput P is obtained. Such bound configuration can be found, for instance, using a binary logarithmic search. The bound b is applied to each interconnection channel f_i where $i \in \{1 \ldots m\}$ for m total channels. A simulation of the program is performed for duration T, and the *peak-capacity*, $Cap_{f_i}^{peak}$ (maximum number of tokens at any given time) of each channel is monitored at every time step. The number of tokens in each channel is expected to fluctuate during simulation due to tokens production and consumption. At the end of simulation at time T, the final values of $Cap_{f_i}^{peak}$ are the minimal buffer size requirements for the throughput P. The new buffer size configuration using this peak capacity is expected to be lower than the original bound b, but is able to guarantee the same execution order and therefore, the same throughput as well. This technique has also been implemented using a TCL script obtaining a fully automatic analysis and optimization tool.

3. BUFFER SIZE OPTIMIZATION BASED ON THE EXECUTION TRACE ANALYSIS

Rather than performing the analysis directly on the DPN network, feasible buffer size configurations can also be found based on the analysis of their executions. For clarity, a short introduction about the concept of execution trace graph is proposed in the following. Interested readers can refers for more details to [13]. The causation trace is a multi directed acyclic graph $G(V, E)$. Each single firing of an action represents a node $\upsilon_i \in V$. An execution dependency between two firings represents an arc (υ_i, υ_j). This latter defines an execution order $\upsilon_i \prec \upsilon_j$, meaning that the execution of υ_j depends on the execution of υ_i. It follows that V can be considered as a partially ordered set of executed actions. Indeed, constructing a consistent dependencies set E is fundamental in order to define constraints on the execution order between any couple of fired actions describing a platform-independent design behaviour. Causation traces need to be built carefully so as to provide a solid basis to produce quantified statements about a dataflow program execution. In fact, in the general case of a dynamic data flow network, such as the one expressed by

the CAL dataflow language, the dependencies set can vary by changing the input stimulus. In other words, the explored states of a dynamic design can be dependent to the provided input. However, in the case of systems implementing several classes of signal processing applications (e.g. video or audio codecs, packet switching in communication networks), probabilistic approaches are meaningful representations of the underlying processing model. Therefore, input sets that are sufficiently *large* with respect to the type of application can be used to generate statistically meaningful causation traces.

Moreover, the causation trace can be analyzed (i.e. post processed) in order to obtain some design performance metrics. For the purpose of this work we make use of the following terms:

- *Weighted execution trace*: the extension of the execution trace obtained by assigning a weight w_{υ_i} to each $\upsilon_i \in \mathrm{V}$.
- *Trace critical path*: the longest weighted path from source to sink nodes in the execution trace. If weights w_{υ_i} correspond to the action firing computational load, then the trace critical path can be related to the overall execution time.

For a given dataflow design it is possible to define a performance upper bound and lower bound in terms of throughput. The highest throughput can be achieved supposing an unbounded or close-to-infinite buffer size configuration. Conversely, the lowest throughput can be obtained with a close-to-minimal buffer size configuration, and consequently the overall memory requirement results to be as well close-to-minimal. The goal is to find a trade-off between design performance (i.e. in terms of throughput) and resource utilization (i.e. in terms of memory necessary to implement all the interconnection buffers).

3.1. Finding close-to-minimum buffer configuration

For a given execution trace, the objective is to find a buffer size configuration x_β such that it guarantees a deadlock-free scheduling with a minimum buffer size configuration. In other terms, we want to find a topological order of V such that all the $\upsilon_i \in \mathrm{V}$ can be executed minimizing the overall buffer size configuration. As proposed in [14], the dependencies set E is split among internal dependencies $\mathrm{E_I}$ and FIFO dependencies $\mathrm{E_F}$. The execution trace graph is then analyzed with a *graph-walk* based method method. At each step k, a single node $\upsilon_i^k \in \mathrm{V}$ is analyzed: If the action is fireable, then it is added to the last position of the new topological sorted vector $\mathbf{V} = \{\upsilon_i \prec \upsilon_{i+1} | \upsilon_i \in \mathrm{V}, \upsilon_{i+1} \in \mathrm{V}\}$, its outgoing FIFO dependencies are made available one by one and consumer nodes are analyzed during the next execution step. If the action is not yet fireable, the graph is walked back to analyze the parent steps. If the dependencies on incoming FIFO are not satisfied, they are polled one by one where the corresponding producer nodes are analyzed during the following execution step. The analysis is completed after all input tokens are consumed, in which case the final action firing schedule is obtained. The buffer size configuration corresponding to this final ordering is found to be close-to-minimum since the execution is ordered such a way that the sequence of action firing generates the smallest amount of token on a given edge.

3.2. Buffer size and throughput trade-off exploration

Given the minimum and maximum buffer size configurations x_β^0 and x_β^{max}, the objective is to find a buffer size configurations with size $B(x_\beta^0) \leq B(x_\beta) \leq B(x_\beta^{max})$ such that a *feasible* trade-off between performance and overall memory requirement is identified. $B(x_\beta^0)$ can be found using one of previous mentioned heuristics; the maximum size is taken as the design with an unbounded buffer size configuration. Consequently, in terms of the trace critical path the following relation can be obtained:

$$CP(x_\beta^{max}) \leq CP(x_\beta) \leq CP(x_\beta^0) \tag{1}$$

Since the trace critical path is defined according to a weighted execution trace, the weights are platform-specific. For a synchronous hardware implementation they correspond to the number of clock cycles required for each action firing. Thence, the weight w_{υ_i} can be defined such that:

$$w_{\upsilon_i} = w_{\upsilon_i}^{\alpha} + w_{\upsilon_i}^{x_\beta} \tag{2}$$

where $w_{\upsilon_i}^{\alpha}$ is the time required for performing the action algorithmic part and $w_{\upsilon_i}^{x_\beta}$ is the time overhead introduced by the finite buffer size configuration x_β. This later models the additional latency incurred by a blocked action firing due to insufficient output buffer space. It represents the time elapsed from the moment υ_i becomes *fireable* until its output buffer has enough space to store the produced tokens. In such case, we can observe that in (1), the trace critical path is longer for buffer configuration x_β compared to x_β^{max} due to $w_{\upsilon_i}^{x_\beta}$ term.

The following describes how the buffer sizes can be incremented for exploring higher throughput. For each exploration step, the size of the most *critical buffer* is increased by the number of maximal blocked tokens $\hat{\tau}(b_i)$. The new buffer configuration is given by

$$x_\beta^{k+1} = x_\beta^k + \hat{\tau}(b_*), b_* \in C_B \tag{3}$$

The set of critical buffers C_B is retrieved from the blocked buffers of the critical actions C_A in the critical path. By increasing the buffer size of the most critical buffer b_*, the overall execution time can be reduced since a higher output rate is obtained due to a larger output buffer space. This process of finding and increasing the critical buffer size at each step can be performed iteratively until: a saturation is reached where

further increments to the critical buffers do not lead to further reductions to the overall execution time or a good trade-off between throughput and overall buffer size requirement is achieved.

4. EXPERIMENTAL RESULTS

Experimental results are applied to the two main components of a standard MPEG-4 AVC/H.264 decoder: the Decoder_Y and the Decoder_U/V branches. Each consists of a residual part with the Hadamard transform and the inverse quantization; the intra prediction part for block sizes of 16x16, 4x4, and 8x8; the inter prediction part with half and quarter pixel interpolation, bilinear interpolation, and the deblocking filter. The number of interconnection channels corresponding to this design is 188 and 58 respectively for Decoder_Y and Decoder_U/V. The throughput, measured in QCIF frames per second (fps), is obtained by simulating the design in RTL to obtain the latency in number of clock cycles per frame. The RTL design is mapped on a Xilinx Virtex-5 FPGA so as to obtain the maximum operating frequency (F_{max} = 114 MHz and 79 MHz respectively for Decoder_Y and Decoder_U/V).

Table 1 summarizes the result of the different dimensioning strategies. Overall we can observe that the direct method (b=8197 for each interconnection) results in very inefficient buffer size requirement. Using the proposed approach for buffer size minimization (TCP_{min}), the total buffer size can be reduced by up to 36 times, but the resulting throughput is also reduced by a factor of about 70%. Compared to the classical Park buffer size minimization approach, our approach also results in a total buffer size reduction of about a factor 2, with only a 16% reduction of throughput. In terms of maximizing the throughput, the proposed approach (TCP_{opt}) results is also quite effective, with a factor 13 reduction of the total buffer size and about 15% throughput reduction when compared to the classical buffer size optimization approach.

The results given in Fig. 1 provides the comparison of the estimated with the actual throughput reduction obtained by applying our buffer size optimization techniques. The analysis has been performed using the TURNUS co-exploration framework developed by the authors [15]. The estimated values are obtained by measuring the total execution time at the dataflow program level, whereas the actual values are obtained by hardware simulation of the synthesized dataflow program. Overall, it can be observed that the estimated and the actual measured throughput values are consistent, and provide very good approximation of the performance gain that can be obtained by actual implementation.

5. CONCLUSION

In this paper, an implementation, validation, and comparison of several buffer size dimensioning techniques applied to DPN implementations has been presented. The proposed approach based on post-processing the execution trace has been compared with a classical state-of-the-art approach. The design case consisted of an FPGA implementation of a very complex processing provided by an MPEG-4 AVC/H.264 decoder employing 246 buffers, a real-size design case that cannot be handled efficiently by direct designer intervention. Results show that our approach is very effective in minimizing and optimizing the total buffer size in a dataflow network, with significant reductions in the required overall size and with the possibility of finding case by case the appropriate trade-offs between buffer resource and streaming throughput, outperforming results obtained using the classical approach.

Table 1. Comparison of total buffer size (Mbits) and throughput (QCIF fps) for all buffer size configuration methods. The design case is on the Decoder_Y and Decoder_U/V of the MPEG-4 AVC/H.264 decoder.

Config method	Decoder_Y		Decoder_U/V	
	Buffer size	Throughput	Buffer size	Throughput
b=8197	19.59	916	4.89	1092
[7, 9](min)	1.12	621	0.31	941
[10, 11](opt)	8.03	916	1.11	1092
TCP_{min}	0.52	534	0.17	909
TCP_{opt}	0.63	797	0.19	1092

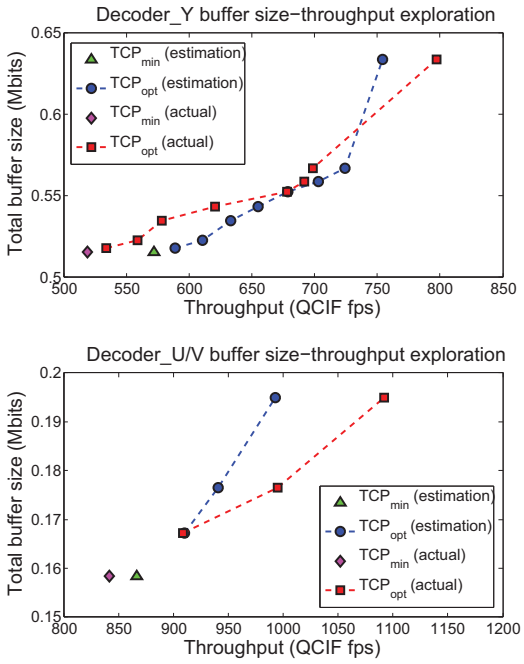

Fig. 1. Buffer size-throughput exploration graphs for estimated and actual performance results using our optimization techniques for the Decoder_Y and Decoder_U/V case studies.

6. REFERENCES

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

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

[3] J. Eker and J. Janneck, *CAL Language Report: Specification of the CAL Actor Language*, University of California-Berkeley, December 2003.

[4] J. Gorin, M. Raulet, Y-L Cheng, H. Y Lin, N. Siret, K. Sugimoto, and G.G. Lee, "An RVC dataflow description of the AVC Constrained Baseline Profile decoder," in *Image Processing (ICIP), 2009 16th IEEE International Conference on*, 2009, pp. 753–756.

[5] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, "Synthesis of embedded software from synchronous dataflow specifications," *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 21, no. 2, pp. 151–166, 1999.

[6] M. Geilen, T. Basten, and S. Stuijk, "Minimising buffer requirements of synchronous dataflow graphs with model checking," in *Proceedings - Design Automation Conference*, 2005, pp. 819–824.

[7] Weichen Liu, Zonghua Gu, Jiang Xu, Yu Wang, and Mingxuan Yuan, "An efficient technique for analysis of minimal buffer requirements of synchronous dataflow graphs with model checking," in *Proceedings of the 7th IEEE/ACM international conference on Hardware/-software codesign and system synthesis*, New York, NY, USA, 2009, pp. 61–70.

[8] S. Casale-Brunet, M. Mattaveli, and J.W. Janneck, "Buffer optimization based on critical path analysis of a dataflow program design," in *IEEE International Symposium on Circuits and Systems 2013 (ISCAS 2013)*, 2013, pp. 1–4.

[9] T. M. Parks, *Bounded Scheduling of Process Networks*, PhD Thesis-University of California-Berkeley, December 1995.

[10] K. Pingali and A. Arvind, "Efficient demand-driven evaluation. part 1," *ACM Transactions in Programming Language Systems*, vol. 7, no. 2, pp. 311–333, April 1985.

[11] K. Pingali and A. Arvind, "Efficient demand-driven evaluation. part 2," *ACM Transactions in Programming Language Systems*, vol. 8, no. 1, pp. 109–139, January 1986.

[12] D. B. MacQueen G. Kahn, "Coroutines and networks of parallel processes," in *Information Processing*, 1977, pp. 993–998.

[13] S. Casale-Brunet, A. Elguindy, E. Bezati, R. Thavot, G. Roquier, M. Mattavelli, and J. W. Janneck, "Methods to explore design space for MPEG RMC codec specifications," *Signal Processing: Image Communication*, vol. 28, no. 10, pp. 1278 – 1294, 2013.

[14] S. Casale-Brunet, C. Alberti, M. Mattavelli, and J. Janneck, "Design space exploration of high-level stream programs on parallel architectures," *Conference: 8th International Symposium on Image and Signal Processing and Analysis (ISPA 2013), Trieste, Italy*, September 2013.

[15] S. Casale-Brunet, C. Alberti, M. Mattavelli, and J. Janneck, "Turnus: a unified dataflow design space exploration framework for heterogeneous parallel systems," *2013 Conference on Design and Archtictures for Signal and Image Processing (DASIP), Cagliari, Italy*, October 2013.