Towards run-time actor mapping of dynamic dataflow programs onto multi-core platforms

Hervé Yviquel, Emmanuel Casseau
University of Rennes 1, IRISA, Inria, France.
firstname.name@irisa.fr

Mickaël Raulet
INSA of Rennes, IETR, France.
firstname.name@insa-rennes.fr

Pekka Jääskeläinen, Jarmo Takala
Tampere University of Technology, Finland.
firstname.name@tut.fi

***Abstract*—The emergence of massively parallel architectures, along with the necessity of new parallel programming models, has revived the interest on dataflow programming due to its ability to express concurrency. Although dynamic dataflow programming can be considered as a flexible approach for the development of scalable applications, there are still some open problems in concern of their execution. In this paper, we propose a low-cost mapping methodology to map dynamic dataflow programs over any multi-core platform. Our approach finds interesting mapping solutions in few milliseconds that makes it doable at regular time by translating it in an equivalent graph partitioning problem. Consequently, a good load balancing over the targeted platform can be maintained even with such unpredictable applications. We conduct experiments across three MPEG video decoders, including one based on the new High Efficiency Video Coding standard. Those dataflow-based video decoders are executed on two different platform: A desktop multi-core processor, and an embedded platform composed of interconnected and tiny Very Long Instruction Word -style processors. Our entire design flow is based on open-source tools. We present the influence of the number of processors on the performance and show that our method obtains a maximum decoding rate for 16 processors.**

I. Introduction

Since processor frequency is bounded due to physics constraints like power dissipation and transistor scaling, multi-core architectures have become the solution to allow performance to keep growing as described by Moore's law. Ongoing embedded platforms, as well as general purpose ones, are containing an increasing number of processor cores, from tens to hundreds. However, these architectures present an interesting, and still open, challenge: the production of applications that fully exploit the parallelism provided by these processors. Contrary to imperative paradigm, dataflow programming paradigm attempts to solve this problem by expressing explicit concurrency within an application. Such a high-level model-based approach enables other nice features such as the re-usability of the components or the dynamic reconfiguration of the application.

Dataflow programming paradigm was used for years to describe streaming applications. Consequently, several kinds of dataflow Models of Computation (MoC) were studied [14]. They can be split into two main classes: the static MoCs allowing a predictable behavior such that scheduling can be done at compile time, and the other MoC having a dynamic behavior, in the sens of data-dependent, such as the Dataflow Process Network (DPN) model [18]. Most of the studies on dataflow programming focus on the statically schedulable MoC because of the efficiency of synthesis techniques on such models. Unfortunately, they do not take into consideration the flexibility and the expressiveness offered to the programmers by the dynamic dataflow MoC. However, such programs need a runtime-scheduling strategy due to their data-dependent behavior.

This paper makes the following contributions:

1) The description of specific criteria involved in the definition of an efficient actor mapping which could be determined at runtime by profiling,
2) The establishment of a dedicated graph partitioning problem that models the actor mapping with a uniform balancing of the application workload onto a multi-core platform to limit the impact on the global performance,
3) We also propose a full mapping methodology to map dataflow programs with unpredictable behavior over any multi-core platform. The simplicity of the method makes it possible in the context of adaptive execution [9].

The paper is organized as follows. First, the context of this work is described in Section II. Then, we explore the related work on actor mapping in Section III. In Section IV we describe our approach to map dynamic dataflow program onto multi-core platforms. Section V presents an experimental evaluation of our actor mapping using well-known video decoders including one based on the new High Efficiency Video Coding (HEVC) standard. Finally, we conclude and consider future works in VI.

II. Dataflow programming

A dataflow program is defined as a graph, as presented in Figure 1, composed of a set of computational units interconnected by communication channels. In the dataflow approach, the communication corresponds to a stream of data composed of a list of tokens.

A. Model of Computation

Dataflow Process Network (DPN) is a model of computation [18] closely related to the well-known KPN [15]: a set of

We would like to thank the organizations which have partially founded this work such as the Center for International Mobility (CIMO) and the Academy of Finland (funding decision 253087).

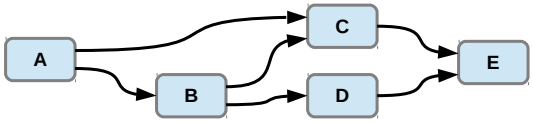

Fig. 1. A dataflow graph

processes, called actors, that communicate with unidirectional and unbounded FIFO channels and connected to ports of actors. Additionally to the KPN model, it introduces the notion of firing. An actor firing, or action, is an indivisible quantum of computation which corresponds to a mapping function of input tokens to output tokens applied repeatedly and sequentially on one or more data streams. This mapping is composed of three ordered and indivisible steps: data reading, then computational procedure, and finally data writing. These functions are guarded by a set of firing rules which specifies when an actor can be fired, i.e. the number and the values of tokens that have to be available on the input ports to fire the actor.

B. Reconfigurable Video Coding

To overcome the lack of interoperability between the several video codecs deployed in the market, MPEG has introduced an innovating framework, called Reconfigurable Video Coding (RVC) [1], dedicated to the development of video coding tools in a modular and reusable fashion. To reach this goal, an RVC codec is described using a domain-specific language, called CAL Actor Language (CAL) [6], which is based upon the DPN model.

In fact, the expressive power of CAL, i.e. its ability to describe *concisely* and *readily* any algorithm, makes it very practical to develop complex applications. However, the poor analyzability of the language at the dataflow-level requires the development of efficient execution techniques to reach the performance needed by real-time execution.

C. Execution model

Since a dataflow program is described as a set of interconnected actors, a multi-core platform could ideally run it by executing each actors in parallel. However, a greater number of actors than processors requires the execution of several actors on the same processor. Fortunately, the strong encapsulation of dataflow components, on top of modeling explicitly the concurrency, lets the choice in a variety of execution models.

A natural approach for handling concurrent execution on a sequential environment is the use of threads, which can lead to a large overhead [4]. Hopefully, DPNs are more suitable than KPNs to be scheduled on sequential environment because no context has to be saved between each actor execution. Indeed, the execution of an actor is described by a sequence of actor firings and actor switchings can only happen between two firings, so only state variables need to be saved. Consequently, it is possible to reduce the overhead of scheduling a dynamic dataflow program by using a user-level scheduler.

In previous work [27], we have presented two efficient dynamic scheduling strategies and have shown that the simplest one, known as *round-robin*, becomes efficient when the number of processors is increasing. Round-robin is a simple scheduling strategy that continuously goes over the list of actors: The scheduler evaluates the firing rules of each actor, executes the actor if a rule is met and continues to execute the same actor until no firing rules are met. This scheduling policy guarantees to each actor an equal chance of being executed, and avoids deadlocks and starvations. Contrary to classical round-robin scheduling, there is no notion of time slice: an actor is executed until it cannot fire anymore because of the states of its FIFO channels in order to minimize the number of actor switchings and consequently the scheduling overhead. Figure 2 shows an application of this round-robin scheduling.

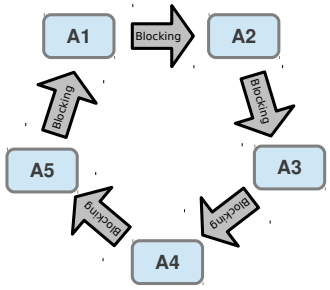

Fig. 2. Example of round-robin scheduling with five actors

Actor composition [12] is another scheduling approach of dynamic dataflow programs which is made possible after the transformation of their firing rules into an original model called actor machine. However, the composition leads to an explosion of the state space of the resulting actors and it is unclear how an embedded platform, hardly constrained in term of memory, can deal with such a consequence.

Wipliez and Raulet [25] present a method to automatically classify dynamic actors into a more restricted dataflow MoC such as synchronous dataflow (SDF) using satisfiability and abstract interpretation. After the classification of the whole program, efficient scheduling techniques can be applied to static or quasi-static regions of the application [11] [2]. Unfortunately, this approach is restricted to local region of the application and no publication have demonstrated yet that a large speedup could be achieved from this method.

Being able to execute several actors on the same processor is not sufficient, we need to insure that the application is equally balanced on the targeted platform to limit the consequence on the performance.

III. Related work

Dataflow-based modeling has always been popular to describe signal processing applications. Consequently, actor mapping and scheduling problematics have been studied by a large number of work. However, most of them [8] [22] stay focus on static dataflow MoC due to the ease to analyze them. Stuijk et al. [23] express the dynamic behavior of an application by describing several static scenarios. Consequently, the programmer has to predict all possible scenarios and describe them in a static way. Shen et al. propose a tool that assists the mapping of a dynamic dataflow program onto an heterogeneous platform [21]. However, they do not address

the issue of automated actor mapping. Schor and al. present a whole scenario-based design flow for mapping streaming application modeled by Kahn Process Networks (KPN) onto on-chip many-core systems [20], but the modeling of the scenarios is not very practical.

Table IV shows a comparison of the approaches reported in literature about the mapping of dataflow applications over multi-core platforms. As can be seen, all of them are based either on static or quasi-static MoC either on KPN and no one is really handling unpredictable behavior. Most of them are automated and executed at runtime but mostly to overcome the overhead of external application or communication mechanism.

TABLE I. COMPARISON OF VARIOUS APPROACHES FOR PERFORMING ACTORS MAPPING OF DATAFLOW PROGRAM OVER MULTI-CORE PLATFORMS

Properties	[22]	[20]	[8]	[23]	[21]	Ours
MoC	SDF	KPN	SDF	SDF	PSDF	DPN
Dynamism	no	scenario	no	scenario	partial	yes
Automatic	yes	yes	yes	yes	directed	yes
Run-time	yes	yes	yes	yes	no	yes
Platform	any	fixed	fixed	fixed	any	any

Since the static analysis of programs based on dynamic dataflow MoC is limited due to their data-dependent behavior, profiling these applications is essential to get some feedback about its efficiency. The widest metric to evaluate this efficiency is the critical path, i.e. the longest, time-weighted sequence of events from the start of the program to its termination. Some works [13] [3] investigate the evaluation of the critical path using post-mortem causation trace at high abstraction-level. One of them [3] has even proposed an algorithm for evaluating the critical path in a linear time. But, contrary to our approach, the need of the causation trace which can be very complex prevents its use at runtime scheduling.

IV. METRICS-BASED ACTOR MAPPING

Since a dataflow program is described as a set of components, called actors, interconnected using FIFO channels, an intuitive way for sharing resources is the execution of several actors on the same processor. However, the actor distribution on the platform should be realized carefully to limit the consequences on the performance.

This section is organized as follow. First, we describe the strategy used to execute several actors on the same processor. Then, we present the metrics which can serve to map efficiently a dynamic dataflow program on a given platform. Finally, we describe the actor mapping into an equivalent graph partitioning problem.

A. Definition of the metrics

We define some metrics about the given application and the targeted platform that help to determine an efficient actor mapping:

- **Resources** are constrained by the targeted platform which contains a limited number of processor cores. If there are less available processors on the platform than actors composing the application, then several actors have to be executed on the same processor.
- Knowing that the **communications** are the common bottleneck of dataflow applications, connectivity between actors is a key factor for mapping them onto the same processor. Reducing the communications between the processors can have two consequences. In case the processors communicate using shared memories, then it limits the pressure on this memory. Or, in case the processors use point-to-point communications, then it simplifies the interconnection network.
- The **performance** limitation of a dataflow program is usually characterized by its critical path. Thereby, we need to reduce the impact of the actor mapping on the critical path and consequently on the global performance. To this end, we define the workload of an actor as the ratio of the computation time in a given time interval as opposed to the time spending to wait for incoming data. The workload of a processor is the sum of the workloads of all its actors plus a small overhead introduced by their scheduling and the limitation of the parallelism. When the workload of a processor overcomes its computation capacity, the critical path is increased because of the global data dependence of a dataflow program.

B. Graph partitioning problem

The mapping of an actor-based application onto a multi-core platform is equivalent to the partitioning of the dataflow graph describing the application.

Given a platform composed of k processors and an application graph $G = (V, E)$ with V a set of vertices modeling the actors and E a set of edges representing the communication channel between the actors. We define $|V|$ the number of vertices and w a function assigning weights to each vertex $v \in V$ such that the weight of a vertex corresponds to the workload of the actor represented by the vertex.

The k-way graph partitioning problem is to partition V into k subsets $V_1, V_2, ..., V_k$, with $V_i \cap V_j = \emptyset$ for $i \neq j$ and $\bigcup_i V_i = V$, such that the sum of the vertex-weights in each subset is balanced and the number of edges whose incident vertices belong to different subsets is minimized. Consequently, the actors will be balanced onto the processors according to their workload in order to minimize the critical path of the whole system. Furthermore, since the algorithm minimizes the edge-cut, the communications between the processors are also minimized.

C. Graph partition method

The graph partitioning problem is NP-complete but some algorithms find high quality partitions in small time using multilevel scheme. A k-way partition is solved by recursive bisection, i.e. the graph is successively split into two balanced partitions. Basically, the bisection is performed using a multilevel algorithm based on the three following phases:

1) **Coarsening phase:** First, the graph G is successively transformed into a series of smaller graphs G_0, .., G_m such that $|V| > |V_0| > ... > |V_m|$. Each of them

is the result of the contraction of some edges from the previous graph. For instance, contracting an edge $(a, b) \in E$ is performed by creating a new vertex $c \in N$ with a weight $w(c)$ equal to the sum of the weight of the edge $w(a, b)$ plus the one of each vertex $w(a)$ and $w(b)$. And, in case the vertice a and b are both connected to the same vertex d, a new edge (c, d) is created such as $w(c, d) = w(a, d) + w(b, d)$.

2) **Partitioning phase:** Then, a bisection of the latest graph G_m is performed, i.e. the coarser graph is partitioned in two balanced partition. Since the reasonable size of the coarser graph, well-known bisection methods such as the Kerninghan-Lin heuristic [17] can find satisfying solutions in a reasonable time.

3) **Uncoarsening phase:** Finally, the resulting partitions are projected back to the original graph by following each transformation made during the coarsening phase, i.e. building successively the equivalent partitions in each intermediate graphs G_0, .., G_m. Between each step, a refinement of the partitioning is performed using the Kerninghan-Lin algorithm [17].

Such an algorithm has been implemented in a tool called Metis [16]. This tools is able to partition a graph with millions vertices in hundreds parts in a few seconds.

D. Mapping flow

Our metrics-based mapping flow maps an application based on a dynamic dataflow model onto a multi-core platform. Since the dynamic behavior of our application makes it unpredictable in most cases, our approach is based on low-cost profiling analysis of the execution. Consequently, we assume that profiling mechanisms are available on the targeted platform.

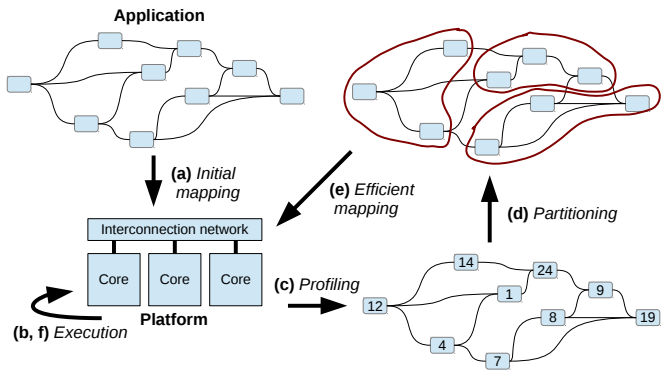

Fig. 3. Metrics-based actor mapping flow

Fig. 3 shows the main steps of our mapping flow. The flow starts from the compilation of the application for the targeted platform using an initial mapping (a), e.g. executing all actors on the same processor core. Then, the application is run on the platform (b) during a predefined time-slice that has to be long enough to represent well the full behavior of the application. After that, the execution profile is analyzed (c) to find out the workload of each actor to get a weighted application graph. Next, the graph partitioning is performed (d) to determine an efficient mapping of the actor onto the available cores of the platform. Finally, the application is recompiled, or reconfigured, according to the computed mapping (e) to enable its efficient execution (f) on the targeted platform.

Finding an optimal mapping of an unpredictable application over a multi-core platform is an NP complete problem. This is why we solve an equivalent graph partitioning problem using reputed heuristics that are able to find an interesting solution in few milliseconds. Moreover, this mapping flow is doable at regular time to keep a good balancing of the application over the processor cores for an efficient processing during the execution of the application.

V. Experimental results

In this section, we present some results about the actors mapping of dynamic dataflow programs over various multi-core platforms. Fig. 4 presents a global view of the compilation flow integrating our automated actor mapping used during these experiments. Our methodology is mainly implemented in Orcc toolset [19] interacting with the external tools such as Metis [16].

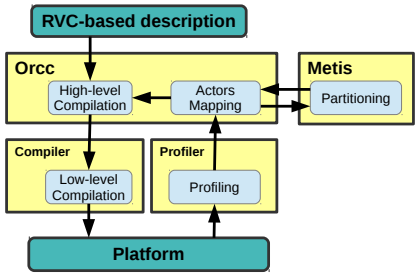

Fig. 4. Compilation flow

The experiments was performed on the dataflow-based implementation of three video decoders developed by the RVC working group: MPEG-4 Part 2 Simple Profile, MPEG-4 Part 10 Progressive High Profile (also known as AVC or H264) and the new MPEG-H Part 2 (better known as HEVC). Table II describes the properties of each description including the profile of the decoder, the parallelization of the luma/chroma decoding, as well as the number of actors and FIFO channels. In fact, the RVC-based video decoders are described with a fine granularity (at block level), contrary to the traditional coarse-grain dataflow (at frame level). This fine-grain streaming approach induces a high potential in pipeline parallelism and the use of small communication channels, between 512 and 8192 for these experiments.

TABLE II. Properties of the description of tested MPEG video decoders

Decoder	Profile	YUV	#Actors	#FIFOs
MPEG-4 Part 2	SP	yes	41	82
AVC / H.264	PHP	yes	114	240
HEVC / H.265	Main	no	38	64

Since the establishment of the first video coding standard, all existing ITU/MPEG video codecs are globally kept the same structure, with an improvement of the algorithmic part to offer an increasing compression rate, that makes the application graph of all RVC-based video codecs quite similar. Figure 5 presents the application graph of the HEVC decoder as a

reference. The graph is decomposed in 4 distinct parts: A first subnetwork, called *parser*, that performs the entropy decoding, which extracts values needed by the next processing from the compressed data stream; Another one, known as *residual*, that decodes the error resulting of the image predication; Following by another one, called *prediction*, that performs the motion compensation. And, a last one dedicated to the post-processing such as the deblocking filer. Each part, expect the parsing, can duplicated for each component (Y, U and V) to increase the data parallelism.

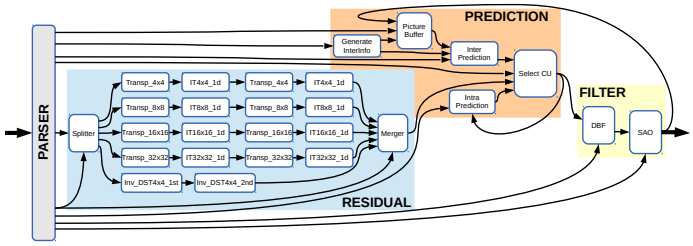

Fig. 5. RVC-based description of the HEVC decoder

A. From desktop multi-core processor

The processor used during this experiments is a Core i7 900 clocked at 3.2GHz. This processor is composed of 6 homogeneous cores sharing 12MB of L3 cache. The processor is executing the compiled source code generating by the software backend (ANSI C) of Orcc. The experiments have been made from the following 720P sequences containing I/P/B frames: Old town cross (25fps and 6Mbps) for MPEG-4 Part 2; A Place at the Table (25fps and 6Mbps) for AVC; Four People (60fps and 1Mbps) for HEVC.

The workload of the actor is evaluated using profiling tools, e.g. Valgrind, during an execution where all actors are mapped to the same core. Since our scheduling strategy is based on the state of the FIFO channels, we assume that the actors are waiting when they are not executed.

TABLE III. SCALABLE FRAME-RATES ON DESKTOP MULTI-CORE PROCESSOR OF MPEG-4 PART 2 (A), AVC (B) AND HEVC (C)

#cores	1	2		3		4		5		6	
(a)	30	54	1.8	76	2.5	93	3.1	90	3	95	3.2
(b)	4	7	1.8	10	2.5	12	3	13	3.3	14	3.5
(c)	7	12	1.7	13	1.9	15	2.1	15	2.1	14	2

The performance obtained with various multi-core configurations using our mapping strategy is presented in Table III. The results show that the maximum decoding rate is obtained using 4 processor cores for all decoders. This limit can be explained by the actor workloads included between 0.1% and 15% of the total execution, i.e. the maximum theoretical speedup would be 6.6x without taking into consideration the cost of communication between the cores, as well as the loop reaction of the data stream between the image buffer and the inter prediction which is a known bottleneck of dataflow-based video decoders. Moreover, The HEVC standard has been designed to take advantage of the increasing parallelism potential of the decoding platforms. However, the dataflow description we use is still experimental and this automated actor mapping will help to its development.

TABLE IV. LIMITS OF HANDMADE PARTITIONING: AN EXAMPLE WITH MPEG-4 PART 2 EXECUTED ON 2 PROCESSOR CORES

Actors mapping	Frame rate	Speed-up
Handmade Luma/Chroma	38 fps	1.26x
Handmade Pipelining	43 fps	1.43x
Automated metrics-based	54 fps	1.8x

Table IV compares the speed-up resulting of the execution of the MPEG-4 Part 2 video decoders over 2 processors using hand-made strategy with our automated method. The results show the difficulty of mapping efficiently by hand a whole dataflow program onto a multiprocessor and prove the interest of an automated method.

B. Towards embedded multi-core platform

The targeted platform is composed of homogeneous Very Long Instruction Word -style processors, based on the Transport-Trigger Architecture [5], and interconnected by point-to-point shared memories. The entire co-design flow used in these experiments [26] has been implemented using two open-source compilers: Orcc [19] and the TTA-based Co-design Environment (TCE) [7] [24].

The workload of each actor is evaluated using a cycle-accurate many-core simulator. In this case, the dataflow application is mapped to an equivalent network of processors, where each actor is executed by its own processor. The time spent in the evaluation of the firing rules can be determined thanks to the call graph generated by the simulator. As expected, most of them spend most of the time waiting input data and do not need an entire processor at full time. The workload of the actors involved in the decoding of the luma is more important due to the chroma sub-sampling.

After profiling, we get some balanced actor mappings of the application on different sized embedded platform. Finally, we evaluate the performance of the resulting partition using our full co-design flow [26]. Figure 6 presents the influence of the number of processors on the performance, in frames per second (FPS), to decode QCIF resolution video based on TTA implementation on a Xilinx Virtex 6 platform (XC6VLX240T) with a 100MHz clock frequency. The form of the curve shows clearly the limit of the task level parallelism of the given application. For MPEG-4 Part 2 SP, the maximum decoding rate is reached with 16 processors. Increasing further the number of processors does not provide higher decoding rate. Our mapping method achieves a speedup ratio of 8.2x with 16 processors in comparison with the single processor execution.

VI. CONCLUSION

In this paper we propose an automated and low-cost metric-based actor mapping of dynamic dataflow programs onto any multi-core platform doable on run-time to overcome the unpredictability of such applications. The mapping is described as an equivalent graph partitioning problem considering the architectural constraints and an earlier profiling which evaluate the weights of each nodes. Then, the graph partitioning problem can be solved using dedicated tools in order to balance the workload of the whole application. The efficiency of our

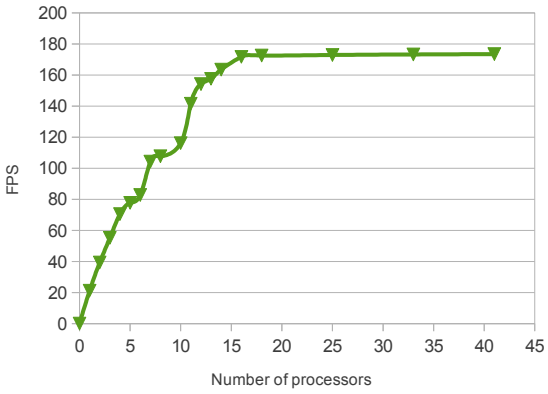

Fig. 6. Scalable frame-rates of an MPEG-4 Part 2 decoder on an embedded multi-core platform

approach is demonstrated by presenting the scalable speed-up of the performance on several well-known video decoders, including one based on the new HEVC standard.

Future works will look at virtual machine technology to enable adaptive execution, such as the Just-in-time Adaptive Decoder Engine [9] [10], that could benefit from this low-cost metric-based actor mapping by offering online actors mapping/scheduling to prove the interest of dynamic dataflow programming languages.

ACKNOWLEDGMENT

We would give special thanks to the Orcc and TCE communities as a whole for actively participating in the development of the tools which offers solid basements to this work.

REFERENCES

[1] S. S. Bhattacharyya, J. Eker, J. W. Janneck, C. Lucarz, M. Mattavelli, and M. Raulet, "Overview of the MPEG Reconfigurable Video Coding Framework," *Journal of Signal Processing Systems*, vol. 63, no. 2, pp. 251–263, Jul. 2009.

[2] J. Boutellier, C. Lucarz, S. Lafond, V. M. Gomez, and M. Mattavelli, "Quasi-static scheduling of CAL actor networks for reconfigurable video coding," *Journal of Signal Processing Systems*, vol. 63, no. 2, pp. 191–202, 2009.

[3] S. C. Brunet, M. Mattavelli, and J. W. Janneck, "Profiling of Dataflow Programs using Post Mortem Causation Traces," in *IEEE Workshop on Signal Processing Systems (SiPS)*. Washington, DC, USA: IEEE Computer Society, 2012, pp. 220–225.

[4] A. Carlsson, J. Eker, T. Olsson, and C. Von Platen, "Scalable parallelism using dataflow programming," *Ericsson Review*, vol. 2, no. 1, pp. 16–21, 2010.

[5] H. Corporaal, *Microprocessor Architectures: from VLIW to TTA*. Chichester, UK: John Wiley & Sons, 1997.

[6] J. Eker and J. W. Janneck, "CAL language report: Specification of the CAL actor language," University of California, Berkeley, Berkeley, Tech. Rep., 2003.

[7] O. Esko, P. Jääskeläinen, P. Huerta, C. S. de La Lama, J. Takala, and J. I. Martinez, "Customized Exposed Datapath Soft-Core Design Flow with Compiler Support," in *Proceedings of the 2010 International Conference on Field Programmable Logic and Applications*. Washington, DC, USA: IEEE Computer Society, Aug. 2010, pp. 217–222.

[8] S. M. Farhad, Y. Ko, B. Burgstaller, and B. Scholz, "Profile-guided deployment of stream programs on multicores," *Proceedings of the 13th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, Tools and Theory for Embedded Systems - LCTES '12*, pp. 79–88, 2012.

[9] J. Gorin, M. Wipliez, F. Prêteux, and M. Raulet, "LLVM-based and scalable MPEG-RVC decoder," *Journal of Real Time Image Processing*, vol. 6, no. 1, pp. 59—-70, 2011.

[10] J. Gorin, H. Yviquel, F. Prêteux, and M. Raulet, "Just-in-time adaptive decoder engine," *Proceedings of the 19th ACM international conference on Multimedia - MM '11*, p. 711, 2011.

[11] R. Gu, J. W. Janneck, M. Raulet, and S. S. Bhattacharyya, "Exploiting Statically Schedulable Regions in Dataflow Programs," *Journal of Signal Processing Systems*, vol. 63, no. 1, pp. 129–142, Jan. 2010.

[12] J. W. Janneck, "A machine model for dataflow actors and its applications," in *Signals, Systems and Computers (ASILOMAR), 2011 Conference Record of the Forty Fifth Asilomar Conference on*. Washington, DC, USA: IEEE Computer Society, Nov. 2011, pp. 756–760.

[13] J. W. Janneck, I. D. Miller, and D. B. Parlour, "Profiling dataflow programs," in *IEEE International Conference on Multimedia and Expo*. Washington, DC, USA: IEEE Computer Society, Jun. 2008, pp. 1065–1068.

[14] W. M. Johnston, J. R. P. Hanna, and R. J. Millar, "Advances in dataflow programming languages," *ACM Computing Surveys*, vol. 36, no. 1, pp. 1–34, Mar. 2004.

[15] G. Kahn, "The semantics of a simple language for parallel programming," *Information processing*, vol. 74, pp. 471–475, 1974.

[16] G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359–392, Jan. 1998.

[17] B. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell system technical journal*, 1970.

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

[19] Orcc, "The Open RVC-CAL Compiler : A development framework for dataflow programs." [Online]. Available: http://orcc.sourceforge.net

[20] 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 *Proceedings of the 2012 international conference on Compilers, architectures and synthesis for embedded systems*. New York, New York, USA: ACM, 2012, pp. 71–80.

[21] C.-C. Shen, H.-H. Wu, N. Sane, W. Plishker, and S. S. Bhattacharyya, "A design tool for efficient mapping of multimedia applications onto heterogeneous platforms," in *IEEE International Conference on Multimedia and Expo (ICME)*. Washington, DC, USA: IEEE Computer Society, 2011, pp. 1–6.

[22] A. K. Singh, A. Kumar, and T. Srikanthan, "A hybrid strategy for mapping multiple throughput-constrained applications on MPSoCs," in *Proceedings of the 14th international conference on Compilers, architectures and synthesis for embedded systems*. New York, New York, USA: ACM, 2011, pp. 175–184.

[23] S. Stuijk, M. Geilen, and T. Basten, "A Predictable Multiprocessor Design Flow for Streaming Applications with Dynamic Behaviour," in *Proceedings of the 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*. Washington, DC, USA: IEEE Computer Society, Sep. 2010, pp. 548–555.

[24] TCE, "The TTA-based Co-design Environment." [Online]. Available: http://tce.cs.tut.fi/

[25] M. Wipliez and M. Raulet, "Classification of Dataflow Actors with Satisfiability and Abstract Interpretation," *International Journal of Embedded and Real-Time Communication Systems*, vol. 3, no. March, pp. 49–69, 2012.

[26] H. Yviquel, J. Boutellier, M. Raulet, and E. Casseau, "Automated design of networks of Transport-Triggered Architecture processors using Dynamic Dataflow Programs," *Signal Processing Image Communication, Special issue on Reconfigurable Video Coding*, 2013.

[27] H. Yviquel, E. Casseau, M. Wipliez, and M. Raulet, "Efficient multicore scheduling of dataflow process networks," in *IEEE Workshop on Signal Processing Systems (SiPS)*. Washington, DC, USA: IEEE Computer Society, 2011, pp. 198–203.