Software synthesis of CAL actors for the MPEG reconfigurable Video Coding framework

Ghislain Roquier, Matthieu Wipliez, Mickael Raulet, Jean François Nezan, Olivier Déforges

SOFTWARE SYNTHESIS OF CAL ACTORS FOR THE MPEG RECONFIGURABLE VIDEO CODING FRAMEWORK

Ghislain Roquier, Matthieu Wipliez, Mickaël Raulet, Jean-François Nezan, Olivier Déforges

IETR laboratory, UMR CNRS 6164, Image and Remote Sensing Group
INSA de Rennes, 20 Avenue des Buttes de Coësmes, 35043 RENNES Cedex, FRANCE
Contacts: {groquier, mwipliez, mraulet, jnezan, odeforge}@insa-rennes.fr

ABSTRACT

The MPEG Reconfigurable Video Coding (RVC) framework aims to provide a unified specification of all video technology. In this framework, a decoder is modularly built as a configuration of video coding tools taken from the MPEG toolbox library. The elements of the library are specified using the CAL Actor Language. CAL is a dataflow based language providing computation models that are concurrent and modular. This paper presents a synthesis tool that from a CAL specification generates multithread C code. Indeed, code generators are fundamental supports for the deployment and success of the MPEG RVC framework. This paper focuses on the automatic translation of a CAL actor. This approach has been used to obtain a C implementation of the inverse DCT module which is a part the MPEG-4 Simple Profile decoder, chosen by MPEG experts to validate the RVC approach. The generated code and the associated translated model are validated against the original CAL description and simulated using the Open Dataflow environment.

Index Terms— MPEG RVC, CAL Actor Language, dataflow modeling, software synthesis

1. INTRODUCTION

A large number of successful MPEG video coding standards have been developed since the first MPEG-1 standard in 1988. The standardization process has always aimed to provide appropriate forms of specifications for a wide and easy deployment. While at the beginning MPEG-1 and MPEG-2 were only specified by textual descriptions, starting with MPEG-4 C/C++ descriptions, reference software became the formal specification of the standard. Such descriptions are composed of non-optimized software packages and face now many limitations. For real implementations all reference SW have to be rewritten to optimize performances and to adapt it to current design methodologies. Such monolithic specifications hide the inherent parallelism and data flow structure of the video coding algorithms, features that are necessary to be exploited for efficient implementations. In the meanwhile the growth of video coding technologies leads to solutions that are increasingly complex to be designed and present significant overlap between standards. Consequently, adding tools in a standard involves a new process in which all the technology is modified. The observation of these drawbacks of current video standard specification formalism led to the development of the Reconfiguration Video Coding (RVC) standard. The key concept was to design a decoder at a higher level of abstraction. An "abstract" model focusing on functionality and concurrency is the specification formalism chosen, which is the best starting point for any design and implementation process. RVC provides a high-level description of the MPEG standard using a specific language called CAL. Once the high-level model is available the challenge is then to develop appropriate tools providing optimized implementations. The work presented here aims to provide a non-normative software synthesis tool called Cal2C. This paper proposes an automated process to handle CAL actors, analyze them and generate multithread C code. The paper is organized as follows: section 2 introduces the RVC framework. This is followed by the description of the CAL actor synthesis process. As part of the MPEG-4 SP decoder, the inverse DCT case study is reported in section 4. Finally conclusions and future work are given in section 5.

2. RVC FRAMEWORK

The MPEG RVC framework is currently under development by MPEG as the part of MPEG-B and MPEG-C standards. It aims to provide a model of defining MPEG standards at system-level [1]. An abstract decoder is built as a block diagram in which blocks define processing entities called Functional Units (FUs) and connections represent the data path. RVC provides both a normative standard library of FUs and a set of decoder descriptions expressed as networks of FUs. Such a representation is modular and helps the reconfiguration of a decoder by modifying the topology of the network. RVC mainly focuses on reusability by allowing decoder descriptions to contain common FUs across standards.

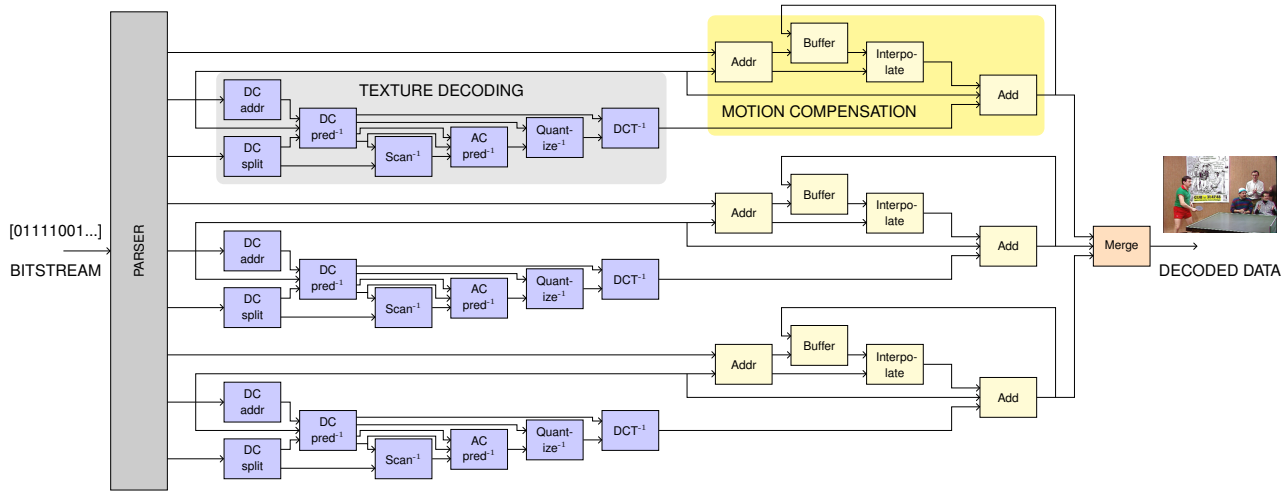

Fig. 1. MPEG-4 Simple Profile decoder description

2.1. FU specification

The CAL Actor Language (CAL) has been chosen as the normative language to specify the standard library of FUs. CAL is a dataflow oriented language created as part of the Ptolemy project [2]. A CAL actor is a computational entity with interfaces (input and output ports), internal state and parameters. An actor is strongly encapsulated; it can neither access nor modify the state of any other actor. An actor may only interact with others by sending data (called tokens) along channels. During an execution (called *firing*), it maps input tokens onto output tokens and changes its internal state. An actor may contain any number of actions that are computations performed during a firing. An action is specified by the number of tokens it consumes and produces. When an actor is fired, it has to select one of them based on the availability of input tokens and optionally based on conditions (called *action guards*) relating to their values or current state. In addition, action selection may be constrained using *action schedules* specified with Finite State Machines (FSM). Action firing is then state-dependent. Finally, *action priorities* impose a partial order among actions to select. In short, an action is fireable if it respects the following conditions: (1) there are sufficient tokens on input ports and adequate room on output ports; (2) guard clauses evaluate to true; (3) the current state enables the action to fire according to the FSM; (4) no higher-priority action respects (1), (2) and (3).

2.2. RVC Decoder Description

The Decoder Description Language (DDL) enables the description of the decoder structure. A decoder is a network formed by a set of interconnected actors. DDL is an XML-based language that can be used to specify the topology of the network. DDL is hierarchical - a network may be a part of a more general network - and is used to pass parameters to actors. For instance, the graphical representation of the macroblock-based MPEG-4 Simple Profile decoder description is shown Figure 1. The parser and the inverse DCT block are hierarchical, otherwise all blocks are atomic actors. Note that only one egde is represented to simplify the decoder when two actors are connected by more than one edge. Decoder descriptions may be generated automatically by a graphical composition tool called Moses[1] that is a CAL editing and simulation framework. The Open Dataflow project[2] is also an environment for building and simulating CAL actor models with a textual description of the network (using the Network Language) that may be automatically translated into an RVC decoder description.

2.3. Network interpretation

The whole system behavior is conditioned by interactions between actors. Those interactions are governed by a Model of Computation (MoC) that defines which scheduling policy is used to execute actors and how communication is done between them. CAL does not have any bias towards any particular MoC. In other words, a CAL network exhibits neither the scheduling policy nor the communication. As a consequence, an RVC decoder description may behave differently depending on the MoC used. This paper does not intend to draw up an exhaustive list of models (cf. [3] for more details). Nonetheless, two dataflow-related MoCs are chosen to demonstrate network interpretation. On one hand, the Process Network (PN) model [4] is efficient when describing asynchronous systems. The PN model is deterministic: the output result is not affected by the scheduling algorithm. However, it requires run-time scheduling, which makes the system verification harder. On the other hand, the Synchronous Dataflow

[1] http://www.tik.ee.ethz.ch/ moses/
[2] http://opendf.sourceforge.net/

(SDF) model is suitable for fixed-rate systems [5]. In SDF, all computation and communication may be statically scheduled. As a consequence, real-time execution and bounded memory usage can be known at compile-time. However, the fixed-rate requirement makes it difficult to model more flexible systems like those in video coding. Indeed, CAL actors in the standard library often contain multiple actions and an abundance of priorities, FSM or guard clauses. It leads to state-dependent or conditional execution hardly modelizable with the SDF formalism. Conversely, the PN model is a model of choice if implementing such networks.

```
actor MUX () int i1, int i2, bool sel ==> int o :
 a1: action i1:[data], sel:[s] ==> o:[data]
  guard s = true
 endaction

 a2: action i2:[data], sel:[s] ==> o:[data]
 endaction

 priority
  a1 > a2;
 endpriority
endactor
```

Fig. 2. if-then-else statement in a basic multiplexer actor

3. CAL ACTOR SYNTHESIS

3.1. Action translation

Actions of an actor are translated to equivalent C functions in several steps. A CAL source file is parsed to an Abstract Syntax Tree (AST) using the rules described in Caltrop Language Report [2]. It holds the same information as CAL source code: both functional and imperative coding styles are used, some variables are missing a type, and actions have different semantics than functions. To generate decent C code, this AST needs to be transformed and fully typed. First, to ease both typing and code generation, actions are considered just as other functions, whose parameters are actions ports. The next step, *type inference*, consists in annotating declarations nodes with type information. To this end, we use the type system along with the $\mathcal{W}$ algorithm defined by Damas and Milner in [6]. This algorithm is able to infer a type-scheme for any expression expressed in the λ-calculus formalism, extended with the *let* clause. A property of this formalism is that any computable function can be expressed using it: each CAL expression is converted to λ-calculus and typed by $\mathcal{W}$. The type information allows other AST transformations that are type-dependent. For the sakes of simplicity and efficiency, CAL lists are modeled as C arrays, requiring list declarations to be updated with computed size information. Another requirement imposed by the target language is that locally-allocated arrays can not be returned on the stack. CAL functions that return a list are thus transformed to take an additional output list parameter instead, and calls to these functions are modified accordingly. A final transformation is replacing purely functional constructs by imperative ones. At this point, the AST is converted to CIL (C Intermediate Language [7]). CIL is a framework and an intermediate language that is both lower-level and more precise than ASTs: it disambiguates C syntactic constructs and embeds type information. This allows seamless translation of expressions and types. C code is generated by calling the pretty-printer included in the framework.

3.2. Guard and priority resolutions

Determining a total order of actions is required to have a consistent evalutation of actions during execution. To this end, considering both action priority and guard may be of help. For instance, an actor with priority and guard is depicted Figure 2. It is a basic multiplexer with 2 actions tagged *a1* and *a2*, 3 input ports *i1*, *i2* and *sel* and 1 output port *o*. During a firing, the actor puts the token from either *i1* or *i2* on *o* according to the boolean value of the token from *sel*. Action tagged *a1* contains a guard clause that constrains the firing. The firing condition for *a1* is the availability of tokens on *i1* and *sel* ports but also that the control token value *s* is true (due to the guard clause) whereas only the availability of input tokens is required for *a2*. Without priority statement, this actor is nondeterministic. Indeed, if the two actions fulfill their firing conditions, it is not possible to determine which action will be fired and then, there is more than one possible output result for the same inputs tokens. To avoid this nondeterminism, action priorities introduce a partial order among the actions that contrain action selection. An action may fire if no higher-priority action is fireable. In the example, *a1* will be selected even if *a2* also fulfill the firing conditions. Formally, the total order is determined as follows; (1) first consider untagged actions, (2) actions without priority statements and finally (3) actions with priority statements. then sort action according to priority if actions have priority statements or according to the document order otherwise.

3.3. Action scheduling

CAL actors may include finite state machine (FSM) to schedule their actions. It consists of a set of states and a set of actions (view as transitions). An action may be fired only if its current state is in accordance with the FSM statement. Figure 3 depicts the graphical representation of the textual FSM inside the ”add” actor from the decoder description. Nodes and edges respectively represent states and actions. This actor adds the prediction error (from the texture decoder) to the prediction (from the motion compensation decoder). Actions schedules constrain action selection according to the current state of actors. For instance, when the current state of ”add” is ”motion”, only *motion* and *done* actions can be evaluated for firing. Formally, FSM translation is done according to the

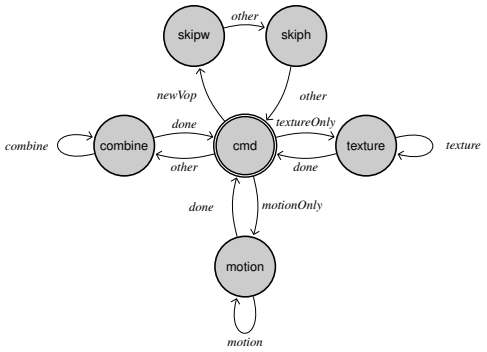

Fig. 3. Graphical representation of the FSM inside the "add" actor

following state-transition mechanisms: (1) each state of the FSM is a label in the C code; (2) the first label in the code is the initial state; (3) each label is followed by function calls to the translated actions (sorted like in 3.2) which are transitions from this state to others; (4) unconditional branch (goto statement) is used after each function call to the next state.

4. RESULTS

The PN model is chosen to interpret CAL networks (cf. section 2.3). Any environments which support multithreading may be chosen to implement PN model. For instance it can be done with POSIX threads by translating CAL actors into threads and by replacing connections with FIFOs. However, low-level considerations such as communication or scheduler implementation, render this solution time-consuming and error prone. Another approach for PN implementation is SystemC[3] [8] whose simulation environment permits high-level programming, well-adapted to functional verifications. A PN-oriented SystemC application is expressed as a network of modules communicating with each other via blocking FIFOs. SystemC comes with an efficient data-sensitive scheduler suitable for dataflow programming. CAL actors are translated into SystemC modules and networks become an instanciation of modules and FIFOs. In order to test the functionality, the case study application is the two-dimensional inverse DCT, which is part of the MPEG-4 SP decoder. More precisely, it is the new normative inverse DCT as specified in [9] and given in Figure 4. It consists of separately applying one-dimensional IDCT along the row and column axis of an 8×8 pixel block. The network is composed of 2 input ports, 1 output port and 7 actors. The testbench consists of applying stimulus (a random number generator) to the top network and verifying the response against an expected result (from the CAL description simulated using the Open Dataflow environment). It has been successfully applied with consistent results between the original and the synthesized networks.

[3]http://www.systemc.org/

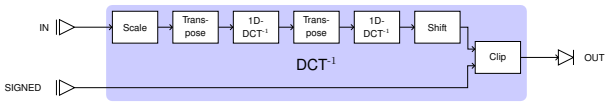

Fig. 4. CAL network of the DCT^{-1} block

5. CONCLUSION AND FUTURE WORK

This paper has presented a brief overview of the process that translates CAL actors into C code. The process has been successfully applied to translate the normative IDCT, as well as most actors inside the MPEG-4 SP decoder description. Moreover, a network implementation has been automatically generated from the top network and simulated using the SystemC environment. The results obtained so far show the efficiency and soundness of the synthezised code. But as far as software is concerned, the PN model faces limitations when it comes to hard real-time implementation. Future work will have to provide network analysis and CAL coding guidelines in order to achieve compile-time scheduling of most MPEG RVC decoder descriptions. More precisely, we plan to automatically transform CAL networks into an SDF-related representation that will allow us to statically schedule and map the resulting model onto multiprocessor architectures.

6. REFERENCES

[1] C. Lucarz, M. Mattavelli, J. Thomas-Kerr, and J. Janneck, "Reconfigurable media coding: a new specification model for multimedia coders," in *Proceedings of SIPS'07*, Oct. 2007.

[2] J. Eker and J. Janneck, "Cal language report," Tech. Rep. ERL Technical Memo UCB/ERL M03/48, University of California at Berkeley, Dec. 2003.

[3] Edward A. Lee, "Embedded software," *Advances in Computers*, vol. 56, pp. 56–97, 2002.

[4] G. Kahn, "The semantics of a simple language for parallel programming," in *Proceedings of IFIP '74*, Aug. 1974, pp. 471–475.

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

[6] Luis Damas and Robin Milner, "Principal type-schemes for functional programs," in *Proceedings of POPL '82*, 1982, pp. 207–212.

[7] G. C. Necula, S. McPeak, S. P. Rahul, and W. Weimer, "CIL: An Infrastructure for C Program Analysis and Transformation," in *Proceedings of CC 2002*, Apr. 2002, pp. 213–228.

[8] Thorsten Grotker, *System Design with SystemC*, Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[9] ISO / IEC 23002-2, "Fixed-point 8x8 IDCT and DCT," 2007.