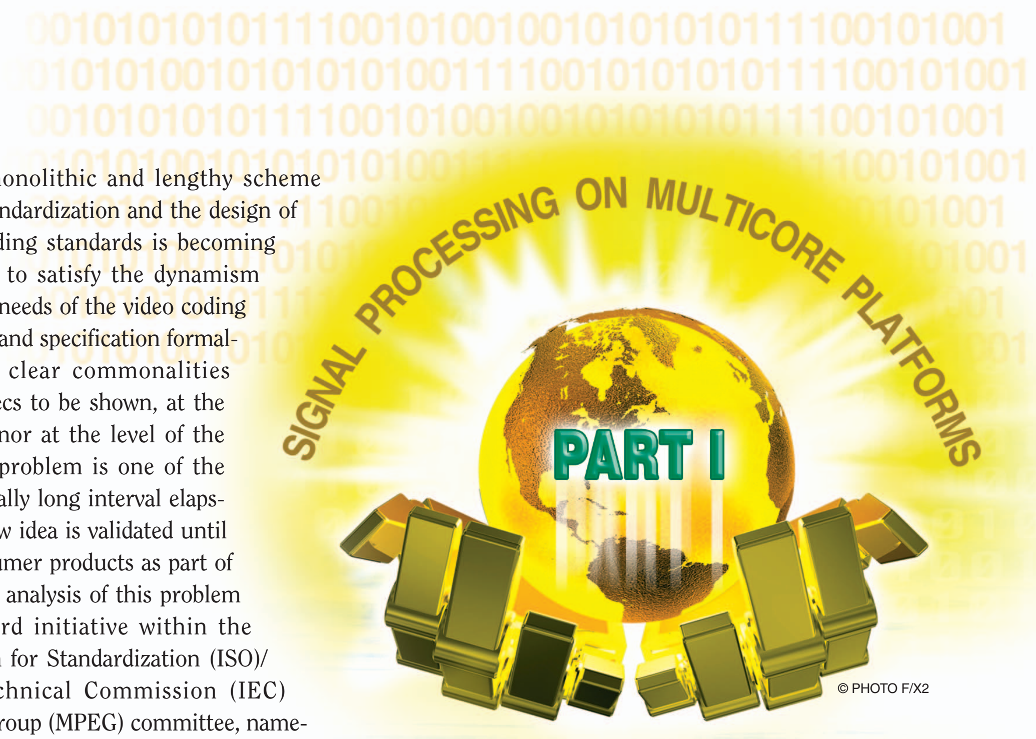
Reconfigurable Video Coding on multicore : an overview of its main objectives

Ihab Amer, Christophe Lucarz, Ghislain Roquier, Marco Mattavelli, Mickael Raulet, Jean François Nezan, Olivier Déforges

Ihab Amer, Christophe Lucarz, Ghislain Roquier,
Marco Mattavelli, Mickaël Raulet, Jean-François Nezan, and Olivier Déforges

Reconfigurable Video Coding on Multicore

An overview of its main objectives



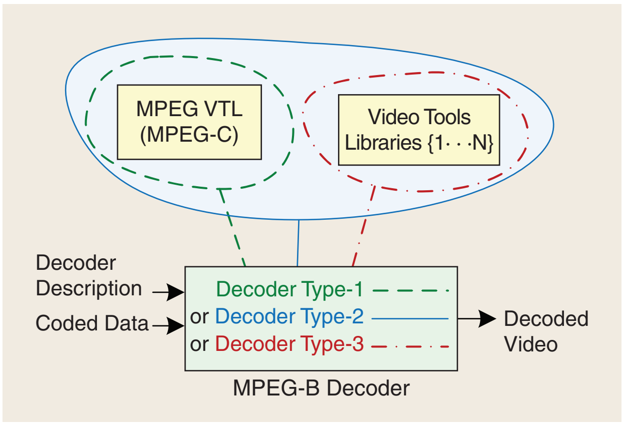
The current monolithic and lengthy scheme behind the standardization and the design of new video coding standards is becoming inappropriate to satisfy the dynamism and changing needs of the video coding community. Such scheme and specification formalism does not allow the clear commonalities between the different codecs to be shown, at the level of the specification nor at the level of the implementation. Such a problem is one of the main reasons for the typically long interval elapsing between the time a new idea is validated until it is implemented in consumer products as part of a worldwide standard. The analysis of this problem originated a new standard initiative within the International Organization for Standardization (ISO)/ International Electrotechnical Commission (IEC) Moving Pictures Experts Group (MPEG) committee, namely Reconfigurable Video Coding (RVC). The main idea is to develop a video coding standard that overcomes many shortcomings of the current standardization and specification process by updating and progressively incrementing a modular library of components. As the name implies, flexibility and reconfigurability are new attractive features of the RVC standard. Besides allowing for the definition of new codec algorithms, such features, as well as the dataflow-based specification formalism, open the way to define video coding standards that expressly target implementations on platforms with multiple cores.

This article provides an overview of the main objectives of the new RVC standard, with an emphasis on the features that enable efficient implementation on platforms with multiple cores. A brief introduction to the methodologies that efficiently map RVC codec specifications to multicore platforms is accompanied with an example of the possible breakthroughs that are expected to occur in the design and deployment of multimedia services on multicore platforms.

Digital Object Identifier 10.1109/MSP.2009.934107

INTRODUCTION

The multicore revolution promises to provide dramatic increases in the performance of processing platforms. However, one of the main obstacles that may prevent the widespread usage of such technology is the fact that current serial specification formalisms and programming methods (the legacy of several years of the continuous successes of the sequential processor architectures) are not at all appropriate to programming the new generation of multicore platforms. Considering the fact that the vast majority of existing software is written in sequential form, and methods such as multithreading rarely scale beyond a few cores, serious problems are arising for porting existing technologies and applications on the new performing multicore platforms. In a time when new hardware meant higher clock frequencies, old programs almost always ran faster on more modern equipment. However, this is not the case anymore when programs written for single-core systems will have to execute on multicore platforms at possibly lower clock speeds on low-power platforms. Hence, a shift to a new programming paradigm that exploits the parallelism and diversification inherited in multicore systems is clearly becoming a necessity. The massive move towards multicore technology is now starting to spread into the multimedia systems world. Many processor manufacturers are working on integrating high definition (HD) encoding/decoding coprocessors with their general purpose CPUs on the same die. More of that is expected to occur in other consumer markets, such as set-top boxes, DVD/Blue-ray players, and mobile phones. With the emergence of the new era of multicore technology, the video coding providers find their way to meet the demands for encoding/decoding platforms capable of supporting multiple coding standards and multiple profiles. Although many of these codecs share common and/or similar coding tools, there is currently no explicit way to exploit such commonalities at the level of the specification nor at the level of implementation. This problem grows as new standards are released and legacy formats continue to be supported, resulting in complex systems that are composed of multiformat algorithms and protocols with virtually unbounded complexities [1], [2]. Ideally, designers of video coding systems should be able to select arbitrary combinations of the available coding tools in a way that the combinations best match the requirements of each application but at the same time such customized applications remain into a standard framework that guarantees interoperability [3]. All of these considerations led to the development of the RVC standard, a new standard currently under its final stage of standardization by the ISO/IEC MPEG, which aims at providing the framework that allows for dynamic development, implementation, and adoption of standardized video coding solutions with features of higher flexibility and reusability [1].



[FIG1] The conceptual view of RVC.

OVERVIEW OF THE MPEG RVC STANDARD

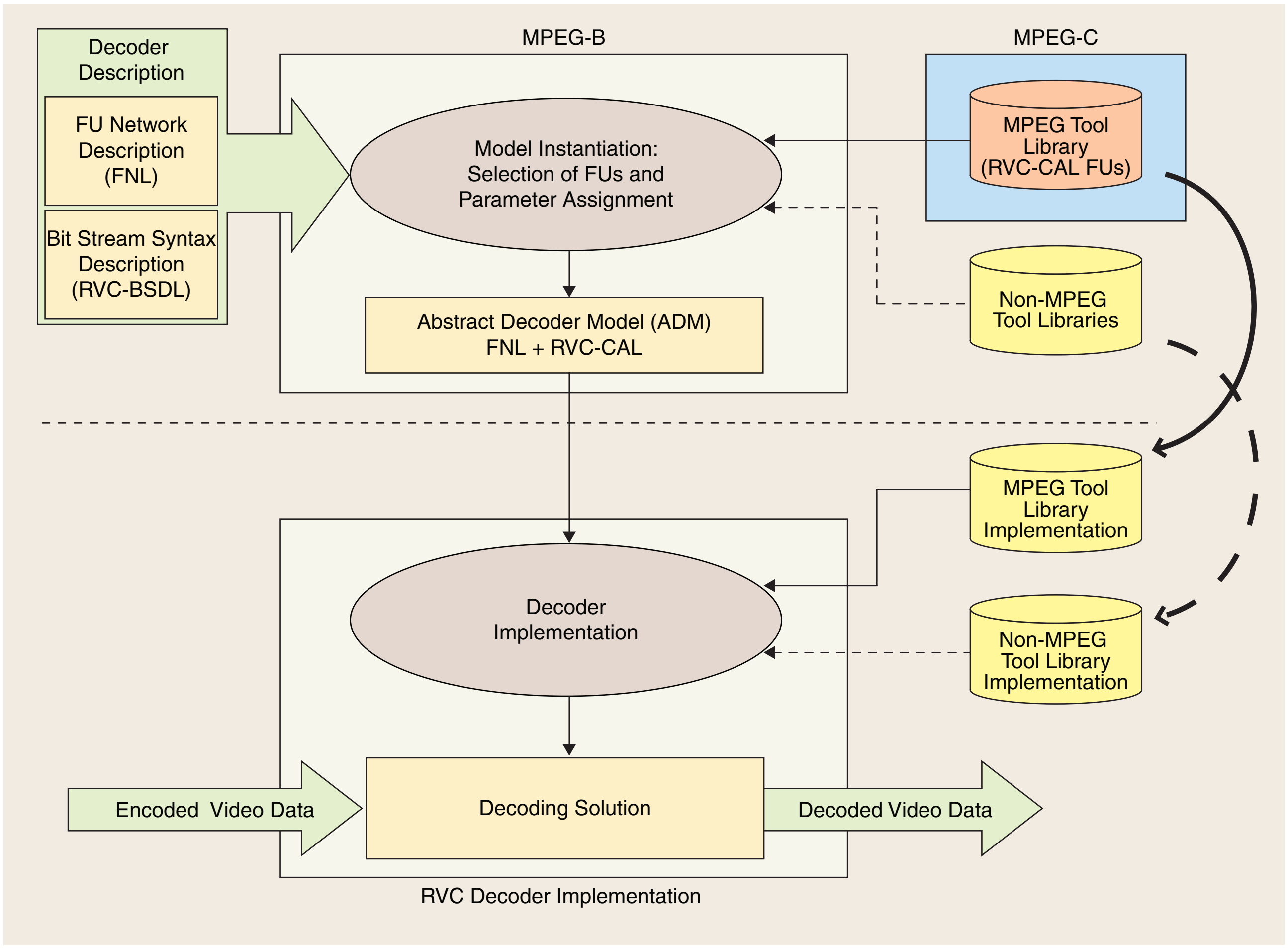
Unlike previous video coding standards, RVC itself does not define a new codec. Instead, it provides a framework to allow service providers to define a multitude of different codecs, by combining together blocks, or so-called functional units (FUs), from a standard video tool library (VTL). Such a possibility clearly simplifies the task of designing future multistandard video coding applications and devices by allowing software and hardware reuse across, once, different, and disjoint video coding standards. Two standards are defined within the context of the MPEG RVC framework: ISO/IEC23001-4 (or MPEG-B part 4) [4], which defines the overall framework as well as the standard languages that are used to describe the different components of the framework, and ISO/IEC23002-4 (or MPEG-C part 4) [5], which defines the library of video coding tools employed in existing MPEG standards [6].

Another interesting feature of the RVC standard that distinguishes it from traditional decoders' rigidly specified video coding standards is that a description of the decoder can be associated to the encoded data in various ways according to each application scenario. Figure 1 illustrates this conceptual view of RVC. All the three types of decoders are within the RVC framework and constructed using the MPEG-B standardized languages. Hence, they all conform to the MPEG-B standard. A Type-1 decoder is constructed using the FUs within the MPEG VTL only. Hence, this type of decoder conforms to both the MPEG-B and MPEG-C standards. A Type-2 decoder is constructed using FUs from the MPEG VTL as well as one or more proprietary libraries (VTL 1-n). This type of decoder conforms to the MPEG-B standard only. Finally, a Type-3 decoder is constructed using one or more proprietary VTL (VTL 1-n), without using the MPEG VTL. This type of decoder also conforms to the MPEG-B standard only. An RVC decoder (i.e., conformant to MPEG-B) is composed of coding tools described in VTLs according to the decoder description. The MPEG VTL is described by MPEG-C. Traditional programming paradigms (monolithic code) are not appropriate for supporting such type of modular framework. A new dataflow-based programming model is thus specified and introduced by MPEG RVC as specification formalism.

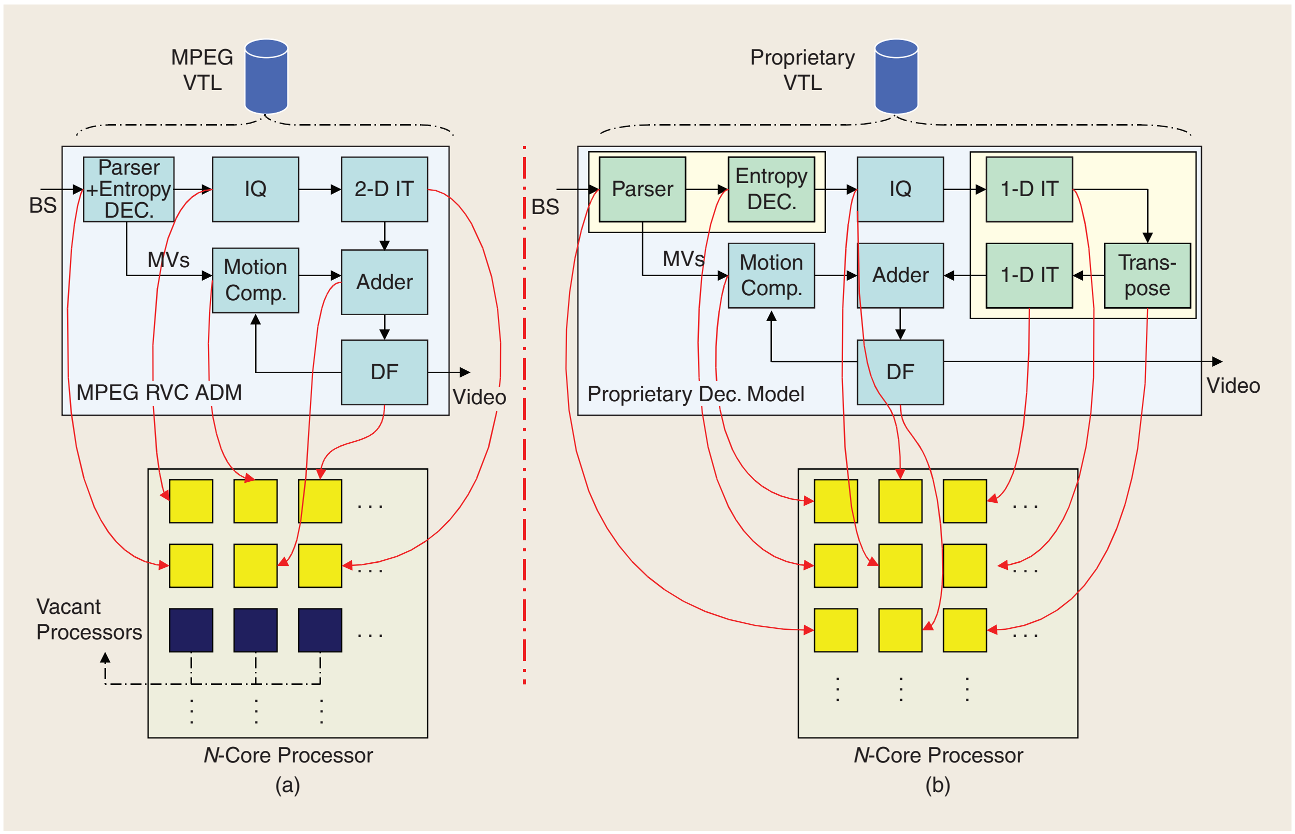
The MPEG VTL is normatively specified using RVC-Caltrop Actor Language (CAL). An appropriate level of granularity for the components of the standard library is important, to enable an effective possibility of reconfigurations, for codecs, and an efficient reuse of components in codecs implementations. If the library is composed of too coarse modules, they will be too large to allow their usage in different and interesting codec configurations, whereas, if the library component granularity level is too fine, the resulting number of modules in the library will be too large for an efficient and practical reconfiguration process at the codec implementation side and may obscure the desired high-level description and modeling features of the RVC codec specifications. Most of the efforts behind the standardization of the MPEG VTL were devoted to study the best granularity tradeoff level of the VTL components. However, it must be noticed that the choice of the best tradeoff in terms of high-level description and module reusability does not really affect the potential parallelism of the algorithm that can be exploited in multicore implementations. This fact becomes clear describing how an RVC implementation is generated and is further developed in the section "Methodologies for Developing RVC Dataflow Programs for Multicore Platforms."

In the RVC framework, the decoding platform acquires the decoder description that fully specifies the architecture of the decoder and the structure of the incoming bit stream. So as to instantiate the corresponding decoder implementation, the platform uses a library of building blocks specified by MPEG-C. Conceptually, such a library is a user-defined proprietary implementation of the MPEG RVC standard library, providing the same input/output (I/O) behavior. This type of library can be expressly developed to explicitly expose an additional level of concurrency and parallelism appropriate for implementing a new decoder configuration on user-specific multicore target platforms. The dataflow form of the standard RVC specification, with the associated model of computation, guarantees that any reconfiguration of the user-defined proprietary library, developed at whatever lower level of granularity, provides an implementation that is consistent with the (abstract) RVC decoder model that is originally specified using the standard library. Figure 2 illustrates the normative and nonnormative components of the RVC framework and shows how a decoding solution is built, not only from the standard specification of the codecs in RVC-CAL by using the normative VTL. This already provides an explicit, concurrent, and parallel model but also from any nonnormative "multicore-friendly" proprietary Video Tool Libraries, that increases if necessary the level of explicit concurrency and parallelism for specific target platforms. Thus, the standard RVC specification that is already an explicit model for multicore systems can be further improved or specialized by proprietary libraries that can be used in the instantiation phase of an RVC codec implementation. In other words, the multicore platform-specific optimization stage is fully integrated into the MPEG RVC standard.

In summary, the new MPEG RVC standard basically provides two levels of explicit concurrency that can be exploited for multicore implementations. The first is at the level of each component of the standard library, that are by definition independent dataflow components, whereas the second is the proprietary customization of each library module that maintains all its internal concurrency properties in whatever configuration they are used. Figure 3 illustrates this concept. The


Decoder Description
FU Network Description (FNL)
Bit Stream Syntax Description (RVC-BSDL)
MPEG-B
Model Instantiation: Selection of FUs and Parameter Assignment
Abstract Decoder Model (ADM) FNL + RVC-CAL
MPEG-C
MPEG Tool Library (RVC-CAL FUs)
Non-MPEG Tool Libraries
MPEG Tool Library Implementation
Non-MPEG Tool Library Implementation
Decoder Implementation
Encoded Video Data
Decoding Solution
Decoded Video Data
RVC Decoder Implementation

[FIG2] Illustration of the reconfigurable video coding framework, showing the dataflow-based normative specification as well as the decoder implementation process that is compatible with the usage of specific "multicore-friendly" libraries.



[FIG3] Illustration of the concept of mapping an RVC specification onto a multicore platform for (a) the abstract decoder model or (b) via user-defined proprietary libraries.

originally instantiated MPEG RVC decoder model that is composed of the basic FUs of the MPEG RVC VTL can already be partitioned and be mapped on a multicore processor platform. In addition, further concurrency can be used if specific proprietary libraries are used leading to higher exploitation of the potential parallelism provided by additional processor cores.

The fundamental element of the RVC framework, in the normative part, is the decoder description that includes two types of data: 1) the bit stream syntax description (BSD), which describes the structure of the bit stream. The BSD is written in RVC-BSD language (BSDL). It is used to generate the appropriate parser to decode the corresponding input encoded data [7], [8]; and 2) the FU network description (FND), which describes the connections between the coding tools (i.e., FUs). It also contains the values of the parameters used for the instantiation of the different FUs composing the decoder [9]–[11]. The FND is written in the so-called FU network language (FNL). The syntax parser (built from the BSD), together with the network of FUs (built from the FND), form a CAL model called the abstract decoder model (ADM), which is the normative behavioral model of the decoder.

Once the ADM is specified, the following step is the implementation of the ADM. As already mentioned above, any proprietary implementation of the standard library, specifically customized for a target multicore platform for instance, can be used for the implementation. Several tools are already available to support such a process, and several others are in development. The more innovative and attractive tools are capable to directly synthesize the ADM into both hardware (HDL) [10] and/or software (C, C++…) [11], [12] implementations using the standard or the proprietary libraries of components. The next sections discuss in more detail how such a process is particularly suitable to provide efficient implementations for multicore platforms. Indeed, the properties of the new specification formalism that is used by the RVC standard provide the right starting point, which is compatible with other advanced methodologies and tools that enable the implementation of RVC codecs on platforms with multiple cores. The dataflow formalism provided by CAL does not imply any specific assumption on the type of multicore architecture (shared memory) or on the topology of their connections, thus any multicore architecture can be the target of the RVC specification. The computation model of the RVC specification implies only autonomous processing entities (actors) exchanging data tokens. It should also be noticed that the implementation process as defined in MPEG RVC provides safe and open ways of reconfiguring a platform without the danger of downloading possibly malicious or dangerous code. In fact, only the codec configuration has to be taken by the outside world, whereas the "pieces of code" that are assembled to build an executable for the final implementation, are either generated directly from the certified standard library, or by the proprietary library that is reliable being built by the user itself or by a trusted source.

CAL DATAFLOW PROGRAMMING MODEL

As mentioned previously, the CAL Actor Dataflow Language has been chosen as the specification language for the behavior of the FUs in the VTL (MPEG-C). CAL was developed and initially specified as a subproject of the Ptolemy project at the University of California at Berkeley [13]. RVC-CAL is a subset of the original CAL language and is normalized by ISO/IEC as a part of the RVC standard. It slightly restricts the data types, operators, and features that could be used in the original CAL language. The main reason for such restrictions is to simplify the development of synthesis tools supporting both hardware and software synthesis [10], [11].

INTRODUCTION TO DATAFLOW PROGRAMMING

The dataflow paradigm for parallel computing has a long history from the early 1970s. Important milestones may be found in the works of Dennis [14] and Kahn [15]. A dataflow program is conceptually represented as a directed graph where nodes (called actors) represent computational units, while edges represent communication channels. These channels are used to send packets of data, called tokens. Unlike the standard imperative programming model, where concurrent programs are typically realized using threads, the dataflow programming model limits the execution of concurrent programs to be only driven by token availability. Low-level considerations, such as synchronization or exclusive memory accesses make the imperative model inadequate in the context of multicore platforms. Conversely, in an actor-oriented dataflow model, an actor executes (fires) when the required token(s) are available, regardless of the status of all other actors. The so-called firing rule (an actor may include several firing rules) defines the amount of tokens to consume and produce and with possibly additional conditions on the token values or the actor state. A major benefit of the dataflow model is that actors may fire simultaneously, thus allowing the program to be distributed over different processing elements, a feature that is particularly useful in the context of multicore platforms.

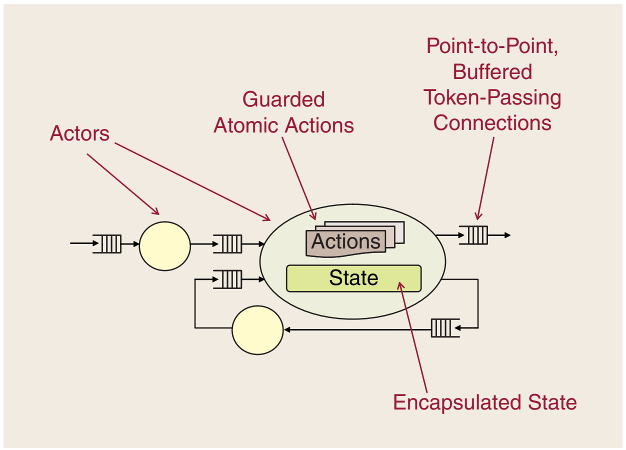
CAL SEMANTICS

Figure 4 illustrates the principles of the CAL dataflow programming model. An actor is a modular component that encapsulates its own state. The state of any actor is not shareable with other actors. Thus, an actor cannot modify the state of another actor. Interactions between actors are only allowed through channels. The behavior of an actor is defined in terms of a set of actions. The operations an action can perform are to consume input tokens, to modify internal state, and/or to produce output tokens. The topology of a set of interconnected actors constitutes what is called a network of actors. The transitions of an actor are purely sequential, where actions are fired one after another. At the network level, the actors can work concurrently, each one executing their own sequential operations. CAL also allows hierarchical system design, in which each actor can be specified as a network of actors.

The CAL dataflow programming model is based on the dataflow process network model (DPN) [16], where an actor may include multiple firing rules. CAL extends the model to cope with nondeterminism. Many variants of dataflow models have been introduced in the literature [17]–[20]. CAL is expressive enough to specify a wide range of programs that follow a variety of dataflow models, trading between expressiveness (the set of programs that can be modeled) and analyzability. A CAL dataflow program can fit into those models depending on the environment and the target application. The synchronous dataflow (SDF) model [17] is one of the most studied in the literature. The SDF model is a special case of DPN where actors have static firing rules. They consume and produce a fixed number of tokens each time they fire. SDF may be easily specified in CAL constraining actions to have the same token consumption and production. Moreover, production and consumption rates may be easily extracted from an actor to check the SDF properties of actors at compile time. A key feature of this model is that a static code analysis detects if the program can be scheduled at compile time. A static analysis produces a static schedule (a predefined sequence of actor firings), if it exists, which is free of deadlock and that uses bounded memory.

WHY C FAILS WHEREAS RVC-CAL MAY WORK FOR MULTICORE PLATFORMS

The control over low-level details, which is considered a merit of C language, typically tends to over-specify programs. Not only the algorithms themselves are specified, but also how inherently parallel computations are sequenced, how and when inputs and outputs are passed between the algorithms and, at a higher level, how computations are mapped to threads, processors, and application specific hardware. In general, it is not possible to recover the original knowledge about the intrinsic properties of the algorithms by means of analysis of the software program. The opportunities for restructuring transformations on imperative sequential code are very limited compared to the parallelization potential available on multicore platforms [21]. These are the main reasons for which C has been replaced by CAL in RVC. Any new codec configuration would have required for a true code parallelization a code analysis that would have resulted to be undecidable without the knowledge of the algorithmic semantic. Knowledge is not available for a new algorithm. Conversely, the CAL language focuses on the



[FIG4] Pictorial representation of the CAL dataflow programming model.

algorithm and does not overload the code with implementation details. Thanks to this property, the CAL language presents interesting features such as parallelism scalability, modularity, flexibility, and portability, as described next.

PARALLELISM SCALABILITY
Writing programs such that their parts execute concurrently without much interference is one of the key problems in scaling traditional imperative programs. Encapsulated actors allow exposing more parallelism as applications grow in size.

MODULARITY
The strong encapsulation of actors along with their hierarchical structure offers high degree of modularity. Hence, the internal specification of any actor can be modified without impacting other actors.

FLEXIBILITY
Unlike procedural programming languages, where control flow (sequence of execution of instructions) is tightly coupled with the algorithm, the actor model allows more flexibility in the scheduling process of the model (i.e., determining the order of execution of the actions) by allowing for various scheduling schemes depending on various optimization criteria.

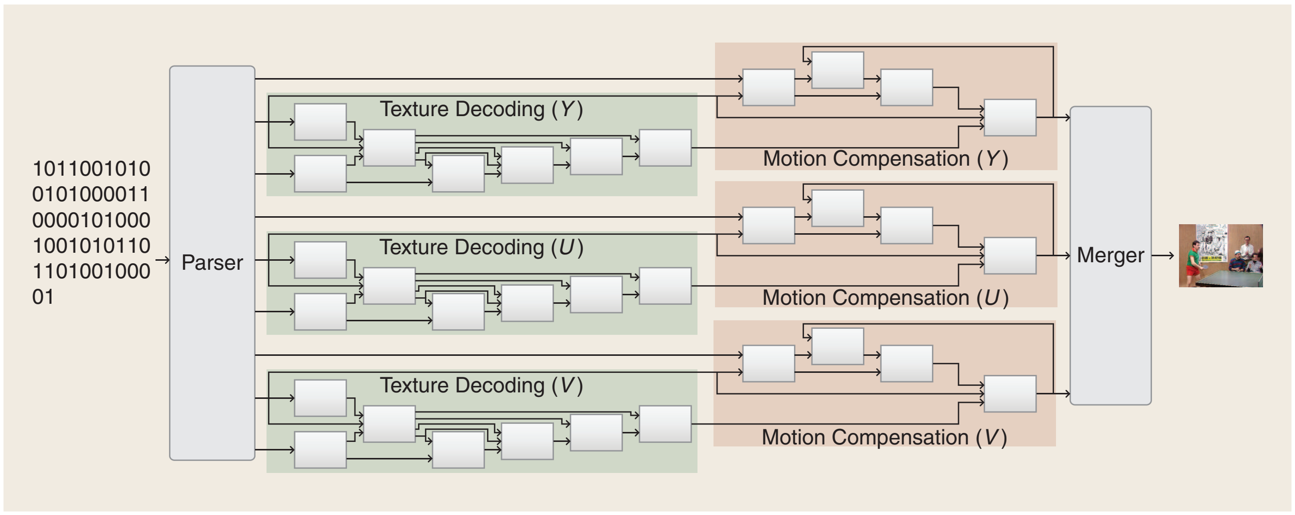
PORTABILITY
For many highly concurrent programs, portability has remained an elusive goal, often due to their sensitivity to timing. The "untimedness" and asynchrony of dataflow programming offers a solution to this problem.

By providing these features, CAL is clearly an appropriate dataflow programming model for modeling and implementing complex signal processing systems on multicore platforms.

IMPLEMENTING RVC DECODERS ONTO MULTICORE PLATFORMS
This section provides two different implementations example of an existing decoder and possible new reconfiguration of RVC decoders targeting a better usage of each specific decoder platform. The first example is a direct synthesis of an ADM over a multicore platform, whereas the second represents a possible RVC implementation of a scalable decoder partitioned over a multicore platform.

AUTOMATIC SYNTHESIS OF AN MPEG-4 SP DECODER ONTO A MULTICORE PLATFORM
The first example is the porting of an RVC MPEG-4 simple profile (SP) decoder. This implementation example intends to show how the RVC specification can directly and automatically generate different model partitions that execute correctly on a parallel platform. Conversely, any mapping of a sequential specification would need a very problematic code analysis to be able to generate parallel code. Thus the emphasis is given to such aspect more than on the absolute performance that implies classical platform specific code optimization steps that can be applied after the dataflow program has been partitioned on different cores and is not the subject of this discussion. Figure 5 presents the structure of the MPEG-4 SP ADM as described within RVC. Essentially, it is composed of four mains parts: the parser, a luminance component (Y) processing path, and two chrominance components (U, V) processing paths. Each path is composed of its texture decoding engine as well as its motion compensation engine (both are hierarchical RVC-CAL FUs).

The potential parallelism of such MPEG-4 SP ADM enables the partitioning of the dataflow model in different ways over a multicore platform. Obviously, the different partitions may result into solutions with different efficiency depending on the combinations of implemented scheduling, data dependencies, and complexities. A software code generator is a very attractive support for deriving implementations from the ADM dataflow model. A code generator, that automatically translates CAL dataflow models to C/C++, is presented in detail in [11]. This tool uses process network model of computation [15] to implement the CAL dataflow model. The compiler creates a multithread program from the given dataflow model, where each actor is



[FIG5] The abtract decoder model of the MPEG-4 SP decoder as defined within RVC.

translated into a thread, and the connectivity between actors is implemented via software first in, first out (FIFO). Although the generation provides correct software implementations, inherent context switches occur during execution, due to the concurrent execution of threads, that may lead to inefficient software execution if the granularity of actor is too fine. The problem of multithreaded programs is discussed in [22]. A more appropriate solution that avoids thread management is presented in [16] and [23]. Instead of suspending and resuming threads based on the blocking read semantic of process network [24], actors are, instead, managed by a user-level scheduler that selects the sequence of actor firing. The scheduler checks (before executing an actor), if it can fire, depending on the availability of tokens on inputs and the availability of rooms on outputs. If the actor can fire, it is executed (these two steps refer to the enabling function and the invoking function of [23]). If the actor cannot fire, the scheduler simply tests the next actor to fire (sorted following an appropriate given strategy) and so on. A code generator based on this concept is available in [25]. Such a compiler presents a scheduler that has the two following characteristics: 1) actor firings are checked at run-time (the dataflow model is not scheduled statically) and 2) the scheduler executes actors following a round-robin strategy (actors are sorted a priori).

In the case of the standard RVC MPEG-4 SP dataflow model, such generated mono-thread implementation is about four times faster than the one obtainable by [11]. For the multicore implementation described here, actors are mapped statically (each actor is assigned a priori to a core) even if nothing in the RVC model prevents implementing for more sophisticated dynamic allocations. For each core, all actors assigned on it are gathered into a single thread managed by its own round-robin scheduler, as explained above. Since only one thread is executed on each core, threads are not executed concurrently but in parallel.

Table 1 shows different implementations with various assignment scenarios of the components of the MPEG-4 SP decoder over a dual-core platform. All of them, and any other partitioning, would execute correctly even if not all of them would be equally meaningful in terms of efficiency.

It is clear that the best achieved frame rate on the target dual-core platform has been obtained by partitioning the decoder in which one core executes the parser in addition to the chrominance engines for texture and motion compensations, while the other core executes the luminance engine for texture/motion compensation in addition to the YUV merging and display engines. This decision has been made knowing that the amount of data processed by the Y channel is typically four times larger than the data processed by each U and V channels (assuming 4:2:0 color resolution). In general, profiling results obtained by simulating the execution of the dataflow models would provide useful indications on how to select appropriate partitioning.

A clear advantage of RVC over the traditional methodology is not only the compactness of the dataflow model (the number of lines of code for the RVC SP decoder used as input of the code generator are only about 3,500) but also the fact that the abstract model of the decoder is platform-independent. This means that, starting from the same MPEG-4 SP ADM, each implementer can realize its own decoder based on the target platform by applying various sets of operations on the ADM so that the resulting implementation is adapted to its own single or multicore target platform. In addition to this platform-independence, the ADM is described in a modular way, allowing for the full exposure of the available data-level parallelism in the decoder.

Table 2 shows an example of the different throughputs that were directly achievable by mapping the MPEG-4 SP ADM to a single and a dual-core platform, respectively. The modularity of the initial ADM, as well as the flexibility provided by the RVC framework and the automatic code synthesis from CAL actors, allowed for a simpler and efficient process to do the mapping on the dual-core platform. Obviously, other platform specific optimizations can be applied to each sequential code section to improve absolute performances. Such classical steps are outside the scope of the discussion and are not presented here.

[TABLE 1] DIFFERENT AUTOMATIC SYNTHESIS OF THE MPEG-4 SP DECODER ON A 2.5 GHZ DUAL-CORE PROCESSOR FOR AN SD SEQUENCE (720 × 576 PIXELS).

CORE 1	CORE 2	FRAMES/S
PARSER + TEXTUREYUV + MOTIONYUV + MERGER		5.9
PARSER + TEXTUREYUV + MOTIONYUV	MERGER	5.7
PARSER + TEXTUREYUV	MOTIONYUV + MERGER	8.5
PARSER + TEXTUREUV + MOTIONUV	TEXTUREY + MOTIONY + MERGER	9.2

RVC PROMISES

The adoption of the MPEG RVC standard is intended to play an important role in renewing many concepts in the field of multimedia systems. One idea that could be applied to video coding to ease the deployment on multicore platforms is to reconfigure a decoder so as to break as much as possible data dependencies. Data dependencies were not considered important factors during standardization in the sequential processor age, however, now they can become the real obstacle for a efficient multicore implementation. Obviously, such a change in algorithms should avoid relevant loss in coding efficiency, and several solutions seem very promising. Previously introduced ideas and technologies such as scalable video coding (SVC) can also be further extended so as to achieve more useful degrees of freedom. Scalability, for instance, could be extended to new dimensions such as the decoder structure itself and/or the required

[TABLE 2] SPEEDUP FACTOR OF THE AUTOMATIC SYNTHESIS OF THE MPEG-4 SP DECODER OVER A 2.5 GHZ DUAL-CORE PROCESSOR.

	THROUGHPUT (FRAMES/S)		SPEEDUP FACTOR
	SINGLE CORE	DUAL CORE	
QCIF (176 × 144)	78.5	122.7	1.56
CIF (352 × 288)	21.4	32	1.49
SD (720 × 576)	5.9	9.2	1.57

processing power. Currently, the objectives on which the state-of-the-art multimedia broadcasting services (such as those based on SVC) rely on, is the rendering of the reconstructed video content in terms of one, or more, of the following three dimensions: quality (SNR), spatial resolution, and/or temporal resolution. This enables the broadcasting services to satisfy the demands of different types of customers having various available transmission bandwidths.

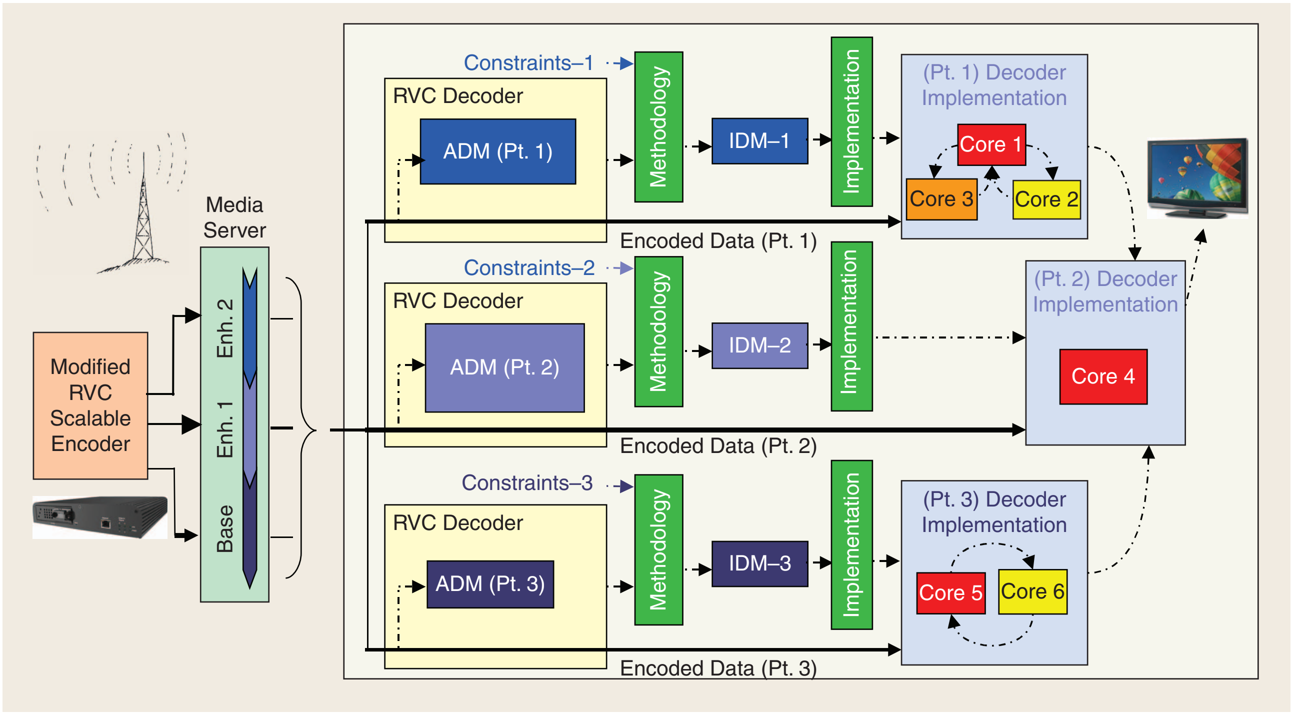
Under the RVC framework, an arbitrary decoder description can be deployed and associated to the encoded bit stream to configure the receiver side on the structure of the decoder. Hence, a theoretically unbounded set of scalability dimensions (other than the three mentioned above) can be formulated by appropriately varying the encoding/decoding scheme, accompanied with embedding the configuration of the new target decoder in the bit stream. Such a possibility extends the potential of better satisfying the demands of target applications to include platform-computational requirements in addition to the bandwidths constraints. Figure 6 provides an example that illustrates such a concept.

An RVC reconfigurable encoder is located at the server side. It produces a scalable bit stream associated with the description of a scalable decoder configuration, in which each enhancement layer contains the enhancement encoded video content, as well as the required data to construct an ADM part that will be able to decode the video content in the corresponding enhancement layer. Hence, unlike the case with the classical SVC, where each decoding terminal has a fixed decoder structure that is able to decode all configurations of the scalable bit stream, the RVC approach is more efficient, enabling for lower usage of computational resources according to the given scalability scenario.

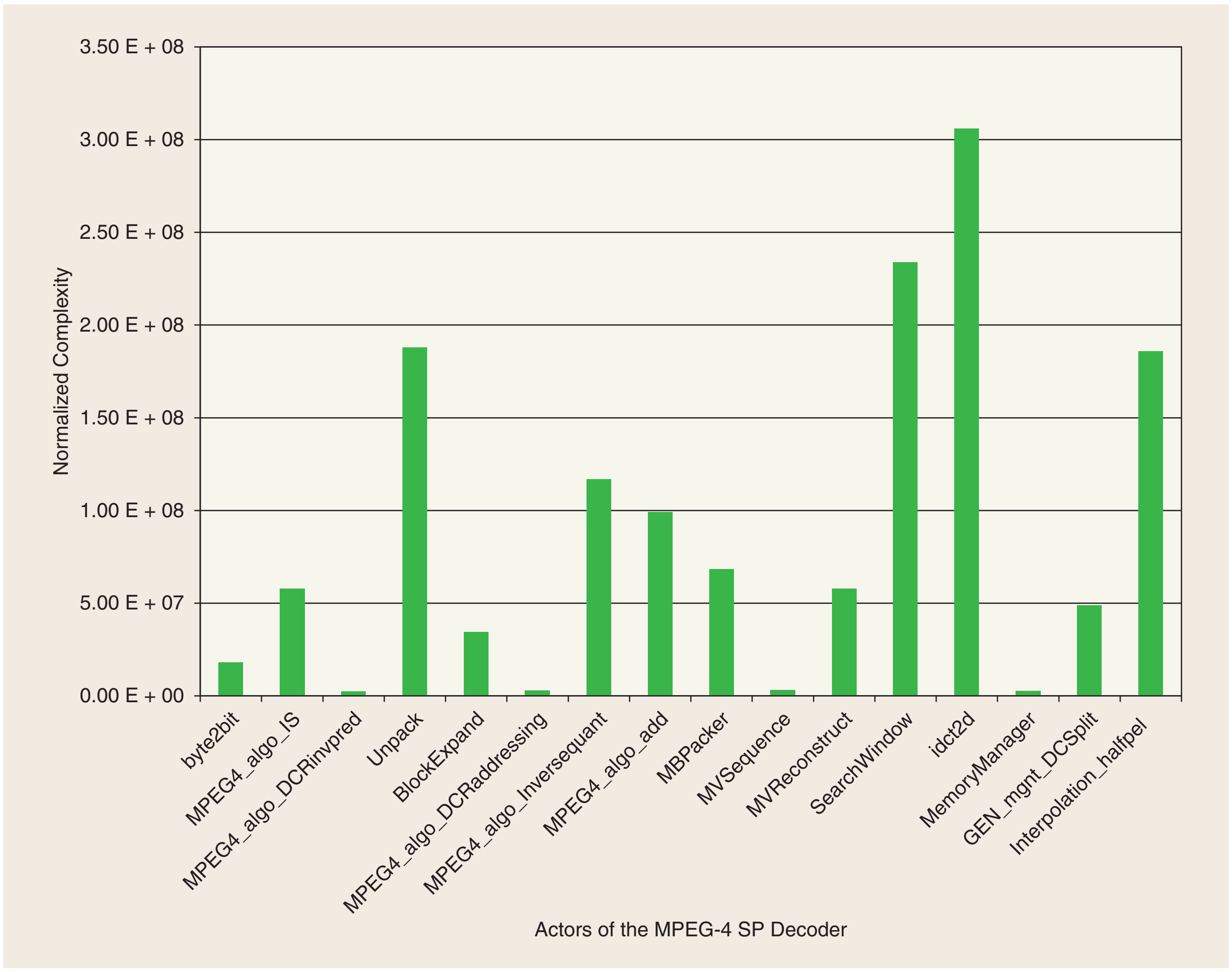
More specifically, a different set of constraints imposed by the target platform can be applied when instantiating the different components of the ADM and that results in different implementation-components that better fits the designated platform(s). This ensures efficient implementation of each abstract decoder part in spite of the differences of the target platforms. Thus, different layer-decoders can be implemented on different platforms, with various core-structures, and then an "integration" core can be used to reconstruct the final video content based on the results of the layered-decoders.

METHODOLOGIES FOR DEVELOPING RVC DATAFLOW PROGRAMS FOR MULTICORE PLATFORMS

The MPEG RVC standard purposely does not specify any methodology for developing a proprietary library of RVC components. This leaves the door open for research and development, either tools for the direct synthesis of a CAL specification into software or hardware or any other methodology for optimization on specific platforms. An example of direct synthesis of an RVC specification has been provided in the previous section. This section presents some ideas and guidelines for the transformation of CAL dataflow programs according to platform constraints for the development of proprietary libraries. Here, the principle is to work at a high level of abstraction and then use tools for direct software synthesis such as the ones mentioned in previous sections. It has to be noticed that nothing prevents using other approaches, for instance, the handwriting of optimized libraries in whatever platform's native language. For the generality of the approach and for the portability on different multicore architectures, our attention and preference



[FIG6] An RVC bit stream can be efficiently decoded by various platforms with various architectures.



[FIG7] Normalized complexity of the action operators of actors composing an MPEG-4 SP decoder.

goes to the development of high-level CAL libraries where RVC CAL components are transformed into hierarchical networks of FUs explicitly exposing higher levels of parallelism. Obviously, the methodology transforming an RVC ADM into a multicore implementation is not limited to developing a library but includes several stages such as partitioning (of components on cores) and scheduling (when more than one FU is partitioned on the same core). In general, such problems are classical problems that are the subject of active research activities and can be solved using different approaches available in literature.

PORTING THE ABSTRACT DECODER MODEL ONTO MULTICORE PLATFORMS

Porting an ADM to fit into a multicore architecture consists of defining where the different actors that compose the ADM will be located during execution. Porting an ADM may require partitioning the ADM where each subset and their corresponding FUs are assigned to a processing element (i.e., core) of the target multicore platform. Once an ADM has been partitioned, each subset is directly translated to C, using the code generator presented in the section "Implementing RVC Decoders Onto Multicore Platforms," regardless of any optimizations. The execution of the model on the different cores is globally asynchronous and is only driven by the token availability, while the actors assigned on a core are scheduled using the aforementioned strategy (c.f. the section "Implementing RVC Decoders Onto Multicore Platforms").

As an example, this concept is illustrated in Figure 3(a), where the different FUs of the MPEG RVC ADM are assigned to the different cores of the platform.

OPTIMIZING THE IMPLEMENTATION AND THE PROPRIETARY TOOLBOXES

The ADM in MPEG is thought to be a very high-level description to ease and represent the video coding process, thus it is mainly concerned with the behavioral description of the different modules of the decoder. An ADM is agnostic of a particular multicore platform. As a consequence, the parallelism and concurrency exposed by the standard ADM generally may not fit well for efficient implementations on a specific target platform.

The proprietary libraries here play a vital role in the process of mapping the ADM to a decoder implementation that meets the constraints of the target platform and the requirements defined by the designer for a specific application. A proprietary toolbox is nonnormative in implementation but normative in a behavior video coding tool library. These FUs must be designed in the proprietary video tool box such that the implementation of the whole decoder by connecting all the FUs as specified by the ADM fulfils the imposed requirements of the system. These proprietary toolboxes intend to provide fine-tuned actors that are considered a benefit for an optimized implementation on a specific multicore architecture.

On one hand, depending on the target multicore platform, the ADM may expose too much parallelism that may lead to an inefficient partitioning of implementation due to the asynchronous execution on the different cores (actors on the same core are scheduled within a round-robin scheduler). Raising the granularity of actors may be a solution to overcome this shortcoming. Subnetworks of actors, (part of the ADM) may be substituted by a single functionally equivalent actor picked from a particular proprietary toolbox. Computations of the different actors are serialized into a single coarse-grain actor so as to increase execution efficiency on each core.

Another option in this context can be directly taken at the ADM level without using any particular proprietary toolboxes. Analyzing the ADM to detect one or more subnetworks that can be scheduled statically (such as a subnetwork with the SDF property) is another possibility. In this case, the resulting scheduling of the actions may be viewed as a larger actor where original actors are executed serially in a predefined order. The resulting partition can thus execute synchronously without needing to check the availability of tokens. The global execution of the model on the different cores is asynchronous where actors assigned to a core execute synchronously. Work is ongoing in this way of exploring the SDF property inside a CAL ADM [26], [27].

In other cases, the ADM may not expose sufficient parallelism for a specific multicore platform. In that case, increasing the potential parallelism so as to exploit the available processing capabilities of the multicore platform is the option to take. For that, a proprietary toolbox may include actors at a finer granularity that can replace actors of the ADM. Such an operation cannot be fully automated because it changes the structure of the dataflow program. Such a model transformation also includes token splitting and the restructuring of the associated actions.

Replacing actors may be done according to various criteria, such as complexity or execution time. Figure 7 illustrates the complexity of the actors that are included in the MPEG-4 SP decoder. The complexity is given in term of elementary (logical, arithmetic, bitwise, etc.) operators, cumulated over the Foreman sequence (QCIF, 300 frames at 30 frames/s). It shows that the inverse transform [a monolithic two-dimensional (2-D) inverse discrete cosine transform (IDCT)] is the most complex part of the decoder in terms of elementary operations. Such analysis shows that if such an actor acts as a bottleneck, it may be replaced by smaller actors provided by a specific proprietary toolbox. For instance, in Figure 3, the 2-D IDCT of the MPEG RVC ADM is replaced with a two one-dimensional IDCTs in the proprietary implementation of the decoder.

CONCLUSIONS

The increasing demand for video decoding platforms capable of supporting multiple codecs, together with the novel ideas that are continuously developing, raises new issues in the standardization process. On one hand, there is the need to release new and more efficient video coding standards in a timely manner to satisfy a wide variety of applications, while on the other hand, being able to guarantee interoperability and support to existing deployed standards is also clearly essential. The lack of such features in current video coding standard specification formalism led to the development of the RVC standard, whose key concept is based on the ability to design a decoder at a higher level of abstraction than the one provided by current generic monolithic specifications. Instead of the typical low-level C/C++ code, an "abstract model," based on modular components taken from the standard library, is the standard reference specification. The possibility of dynamic reconfiguration of codecs also requires new methodologies and new tools for describing the new bit stream syntaxes and the parsers of such new codecs. The functionality of the coding tools and their potential concurrency are explicitly exposed to the users by the used specification formalism. This new approach to the codec specification is by far a better starting point, not only for customizing audio-visual applications to the specific service requirements, but also for efficient implementation methodologies on the new generation of multicore platforms. With the emergence of the RVC standard and its supporting technologies and tools, many concepts in the field of multimedia systems and their implementations on multicore platforms are expected to be revisited and evolve into new solutions in the future.

ACKNOWLEDGMENTS

The authors would like to thank Matthieu Wipliez, Jérôme Gorin, and Mickaël Raulet, who developed the Open RVC-CAL Compiler framework, and Jörn W. Janneck and Johan Eker, of the Open Dataflow framework.

AUTHORS

Ihab Amer (ihab.amer@epfl.ch) received his B.Sc. degree in 2000 and his M.Sc. degree in 2003. He received his Ph.D. degree from the University of Calgary, Canada, in 2007. He spent six months with the DSP division of Xilinx Inc., and he has been a member of the ISO/IEC MPEG Standard Committee. He authored/coauthored over 30 peer-reviewed conference and journal papers, and he has over 30 contributions to the MPEG standards. He also has a pending patent. He is an assistant professor at the German University in Cairo, a research consultant with ATIPS Labs at the University of Calgary, and a visiting researcher at the GR-LSM Lab at Ecole Polytechnique Fédérale de Lausanne (EPFL-Switzerland). He is a Member of the IEEE.

Christophe Lucarz (christophe.lucarz@epfl.ch) received his M.Sc. degree in electrical engineering from the Institut National des Sciences Appliquées in Lyon, France, in 2006. He is currently a researcher with the Multimedia Architectures Research Group at Ecole Polytechnique Fédérale de Lausanne (EPFL-Switzerland) and is working towards his Ph.D. degree. He is also taking part in the MPEG ISO/IEC standardization committee in video coding as well as the ACTORS European Project.

Ghislain Roquier (ghislain.roquier@epfl.ch) is a post-doc researcher in the GR-LSM Multimedia group at Ecole Polytechnique Fédérale de Lausanne in Switzerland. He received his M.Sc. degree in signal processing from the Université de Rennes I, France in 2005. He received the Ph.D. degree in electronics in 2008 from the Institut National des Sciences Appliquées de Rennes, France. His main research interests include design and implementation of heterogeneous real-time embedded systems, rapid prototyping methodologies, and multimedia DSP.

Marco Mattavelli (marco.mattavelli@epfl.ch) started his research activity at the "Philips Research Laboratories" of Eindhoven in 1988. In 1991 he joined the "Swiss Federal Institute of Technology" (EPFL), where he got his Ph.D. in 1996. He received the ISO/IEC Award in 1997 and 2003. He is currently leading the "Multimedia Architectures Research Group" at EPFL. His current major research activities include methodologies for specification and modeling of complex systems and architectures for video coding. He is the author of more than 100 publications and has served as invited editor for several conferences and scientific journals. He is a Member of the IEEE.

Mickaël Raulet (mraulet@insa-rennes.fr) received his M.Sc. degree in 2002. He received a Ph.D. degree from the Institut National des Sciences Appliquées (INSA) in electronic and signal processing in collaboration with the software radio team of Mitsubishi Electric ITE (Rennes, France) in 2006. He is currently with the Institute of Electronics and Telecommunications of Rennes, where he is a researcher in rapid prototyping of standard video compression on embedded architectures. Since 2007, he has been involved in the ISO/IEC JTC1/SC29/WG11 standardization activities (better known as MPEG) such as a Reconfigurable Video Coding Expert. His interests include video standard compression and telecommunication algorithms and rapid prototyping onto multicore architectures. He is a Member of the IEEE.

Jean-François Nezan (jnezan@insa-rennes.fr) is an assistant professor at the Institut National des Sciences Appliquées (INSA) in Rennes, France, and a member of the IETR Laboratory also in Rennes. He received his postgraduate certificate in signal, telecommunications, images, and radar sciences from Rennes University in 1999, and his engineering degree in electronic and computer engineering from INSA-Rennes Scientific and Technical University in 1999. He received his Ph.D. degree in electronics in 2002 from INSA. His main research interests include image compression algorithms, multiprocessor rapid prototyping, and dataflow programming.

Olivier Déforges (odeforge@insa-rennes.fr) is a professor at the Institut National des Sciences Appliquées (INSA) in Rennes, France. He received a Ph.D. degree in image processing in 1995. In 1996, he joined the Department of Electronic Engineering at INSA-Rennes Scientific and Technical University. He is a member of the Institute of Electronics and Telecommunications of Rennes, UMR CNRS 6164. His research interests are image and video lossy and lossless compression, image understanding, fast prototyping, and parallel architectures.

REFERENCES

[1] E. S. Jang, J. Ohm, and M. Mattavelli. (2008). *Whitepaper on reconfigurable video coding (RVC)*. ISO/IEC JTC1/SC29/WG11 Document N9586, Antalya, Turkey [Online]. Available: http://www.chiariglione.org/mpeg/technologies/mpb-rvc/index.htm

[2] I. Richardson, C.S. Kannangara, M. Bystrom, J. Philp, and M. De. Frutos-Lopez, "A framework for fully configurable video coding," in *Proc. Picture Coding Symp. (PCS)*, 2009, pp. 145–148.

[3] C. Lucarz, M. Mattavelli, J. Thomas-Kerr, and J. Janneck, "Reconfigurable media coding: A new specification model for multimedia coders," in *Proc. IEEE Workshop Signal Processing Systems (SiPS)*, 2007, pp. 481–486.

[4] *MPEG Systems Technologies—Part 4: Codec Configuration Representation*, ISO/IEC FDIS 23001-4, 2009.

[5] *MPEG Video Technologies—Part 4: Video Tool Library*, ISO/IEC FDIS 23002-4, 2009.

[6] C. Lucarz, I. Amer, and M. Mattavelli, "Reconfigurable video coding: Concepts and technologies," in *Proc. IEEE Int. Conf. Image Processing, Special Session on Reconfigurable Video Coding*, Cairo, Egypt, 2009.

[7] *MPEG Systems Technologies—Part 5: Bitstream Syntax Description Language (BSDL)*, International Standard ISO/IEC FDIS 23001-5, 2008.

[8] M. Raulet, J. Piat, C. Lucarz, and M. Mattavelli, "Validation of bitstream syntax and synthesis of parsers in the MPEG reconfigurable video coding framework," in *Proc. IEEE Workshop Signal Processing Systems (SIPS)*, 2008, pp. 293–298.

[9] D. Ding, L. Yu, C. Lucarz, and M. Mattavelli, "Video decoder reconfigurations and AVS extensions in the new MPEG reconfigurable video coding framework," in *Proc. IEEE Workshop Signal Processing Systems (SIPS)*, 2008, pp. 164–169.

[10] J. W. Janneck, I. D. Miller, D. B. Parlour, G. Roquier, M. Wipliez, and M. Raulet, "Synthesizing hardware from dataflow programs: An MPEG-4 simple profile decoder case study," in *Proc. IEEE Workshop Signal Processing Systems (SIPS)*, 2008, pp. 287–292.

[11] G. Roquier, M. Wipliez, M. Raulet, J. W. Janneck, I. D. Miller, and D. B. Parlour, "Automatic software synthesis of dataflow program: An MPEG-4 simple profile decoder case study," in *Proc. IEEE Workshop Signal Processing Systems (SIPS)*, 2008, pp. 281–286.

[12] G. Roquier, C. Lucarz, M. Mattavelli, M. Wipliez, M. Raulet, J. W. Janneck, I. D. Miller, and D. B. Parlour, "An integrated environment for HW/SW co-design based on a CAL specification and HW/SW code generators," in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS 2009)*, Taipei, Taiwan, 2009, p. 799.

[13] J. Eker and J. W. Janneck. (2003). *CAL language report specification of the CAL actor language*. EECS Dept., Univ. of California, Berkeley, Tech. Rep. UCB/ERL M03/48 [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2003/4186.html

[14] J. B. Dennis, "First version of a data flow procedure language," in *Proc. Colloque sur la Programmation Programming Symp*. London, U.K.: Springer-Verlag, 1974, pp. 362–376.

[15] G. Kahn, "The semantics of a simple language for parallel programming," in *Information Processing*, J. L. Rosenfeld, Ed. Stockholm, Sweden: North Holland, Aug. 1974, pp. 471–475.

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

[17] E. Lee and D. Messerschmitt, "Synchronous data flow," *Proc. IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[18] S. Ritz, M. Pankert, V. Zivojinovic, and H. Meyr, "Optimum vectorization of scalable synchronous dataflow graphs," in *Proc. IEEE Int. Conf. Application-Specific Array Processors,* Oct. 1993, pp. 285–296.

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

[20] B. Bhattacharya and S. Bhattacharyya, "Parameterized dataflow modeling for DSP systems," *IEEE Trans. Signal Process.*, vol. 49, no. 10, pp. 2408–2421, 2001.

[21] S. S. Bhattacharyya, G. Brebner, J. Eker, J. W. Janneck, M. Mattavelli, C. von Platen, and M. Raulet, "OpenDF—A dataflow toolset for reconfigurable hardware and multicore systems," *ACM SIGARCH Comput. Architecure News, Special Issue: MCC08–Multicore Computing 2008*, vol. 36, no. 5, pp. 29–35, Dec. 2008.

[22] E. A. Lee, "The problem with threads," *IEEE Comput. Soc.*, vol. 39, no. 5, pp. 33–42, 2006.

[23] W. Plishker, N. Sane, M. Kiemb, K. Anand, and S. S. Bhattacharyya, "Functional DIF for rapid prototyping," in *Proc. 2008 19th IEEE/IFIP Int. Symp. Rapid System Prototyping*, June 2008, pp. 17–23.

[24] G. Kahn and D. B. MacQueen, "Coroutines and networks of parallel processes," in *Proc. IFIP Congr.*, 1977, pp. 993–998.

[25] *Open RVC-CAL Compiler* [Online]. Available: http://sourceforge.net/projects/orcc/

[26] J. Boutellier, V. Sadhanala, C. Lucarz, P. Brisk, and M. Mattavelli, "Scheduling of dataflow models within the reconfigurable video coding framework," in *Proc. IEEE Workshop Signal Processing Systems (SIPS)*, 2008, pp. 182–187.

[27] R. Gu, J. Janneck, M. Raulet, and S. S. Bhattacharyya, "Exploiting statically schedulable regions in dataflow programs," in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing*, Taipei, Taiwan, Apr. 2009, pp. 565–568. [SP]