Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2012, Article ID 484962, 11 pages
doi:10.1155/2012/484962

Research Article

Hardware and Software Synthesis of Heterogeneous Systems from Dataflow Programs

Ghislain Roquier, Endri Bezati, and Marco Mattavelli

Ecole Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland

Correspondence should be addressed to Ghislain Roquier, ghislain.roquier@epfl.ch

Received 15 July 2011; Revised 27 October 2011; Accepted 6 December 2011

Academic Editor: Deming Chen

The new generation of multicore processors and reconfigurable hardware platforms provides a dramatic increase of the available parallelism and processing capabilities. However, one obstacle for exploiting all the promises of such platforms is deeply rooted in sequential thinking. The sequential programming model does not naturally expose potential parallelism that effectively permits to build parallel applications that can be efficiently mapped on different kind of platforms. A shift of paradigm is necessary at all levels of application development to yield portable and scalable implementations on the widest range of heterogeneous platforms. This paper presents a design flow for the hardware and software synthesis of heterogeneous systems allowing to automatically generate hardware and software components as well as appropriate interfaces, from a unique high-level description of the application, based on the dataflow paradigm, running onto heterogeneous architectures composed by reconfigurable hardware units and multicore processors. Experimental results based on the implementation of several video coding algorithms onto heterogeneous platforms are also provided to show the effectiveness of the approach both in terms of portability and scalability.

1. Introduction

Parallelism is becoming more and more a necessary property for implementations running on nowadays computing platforms including multicore processors and FPGA units. However, one of the main obstacles that may prevent the efficient usage of heterogeneous platforms is the fact that the traditional sequential specification formalisms and all existing software and IPs, legacy of several years of the continuous successes of the sequential processor architectures, are not the most appropriate starting point to program such parallel platforms [1]. Moreover, such specifications are no more appropriate as unified specifications when targeting both processors and reconfigurable hardware components. Another problem is that portability of applications on different platforms becomes a crucial issue and such property is not appropriately supported by the traditional sequential specification model and associated methodologies. The work presented in this paper focuses in particular on a methodology for the generation of scalable parallel applications that provide a high degree of portability onto a wide range of heterogeneous platforms. We argue that to achieve such objectives is necessary to move away from the traditional programming paradigms and adopt a dataflow programming paradigm. Indeed, dataflow programming explicitly exposes the parallelism of applications, which can then be used to distribute computations according to the available parallelism of the target platforms. Moreover, the methodology described here has also the objective of raising the level of abstraction at all levels of the design stages involving human intervention for facilitating the overall design of complex applications onto heterogeneous systems, composed of multicore processors and FPGAs.

A key requirement in our design approach is that applications have to be portable and scalable. Portability ensures fast deployment of applications with minimal assumption on the underlying architecture, which drastically shortens the path from specification to implementation. The application should be able to run on any processing component architecture of a heterogeneous system from a single description, without code rewriting. Another important feature is that applications should also be scalable. It means that the

performance of the running application should scale with the available parallelism of the target architecture.

The following sections present the main stages of a dataflow-based approach that present the described features in the design of applications on heterogeneous platforms.

2. Related Works

Hardware-Software (HW-SW) codesign concept and fundamental ideas, that are also at the base of our work, have been introduced in the nineties [2, 3]. A formal definition of the term codesign is not unique. In the rest of the document codesign stands for the joint design of SW and HW components from a single-application description.

Our application model is in the line with model-based design. Model-based design was proposed to raise the level of abstraction when designing digital processing systems. High-level models provide useful abstractions that hide low-level details, such as platform independency, in order to ease analysis tasks. Prior research related to model-based design using data- and control-dominated models and a combination of both is the subject of a wide literature. Essentially the various methods proposed mainly differ by the model used and by the so-called model of computation (MoC).

Without claiming to be exhaustive, we can mention the POLIS [4] framework based on Codesign Finite State Machine (CFSM) that relaxes FSMs to communicate asynchronously. Such model has limited expressive power which is a key feature when targeting the design of critical reactive systems and results rather difficult to be used outside the scope of control-dominated systems.

By contrast to control-dominated models, data-dominated models such as communicating sequential processes (CSPs), dataflow graphs or Kahn process networks (KPNs) are preferred when dealing with stream processing algorithms.

SynDEx from INRIA [5] is one of such design approaches based on a restricted dataflow model. In this model, a vertex of the dataflow graph may be factorized (n-times repetitions during the graph execution) and conditioned (conditional statements by using the hierarchy in vertices and control values). The SynDEx model is deliberately restricted to ensure real-time constraints and consequently is not always adapted to model more general class of applications. Moreover, the high-level synthesis (HLS), that turns the model to HDL, is no more maintained.

Compaan/Laura from Leiden University [6] is based on KPN approach by using a subset of MATLAB code to model applications. The HW back-end (Laura) then turns the KPN model expressed by MATLAB code to VHDL. KPN-based models are much more expressive than more restricted MoC and can cover a much broader class of applications. However, since analyzability is, roughly speaking, inversely related to the expressiveness, it is somehow difficult to figure out the ability to generate the corresponding KPN models of more complex applications written in MATLAB.

PeaCE from the Seoul National University is an approach that lays at midway between dataflow (synchronous dataflow—SDF) and FSM [7]. This model raises the level of expressiveness, by enabling the usage of more control structures using FSM inside SDF vertices and vice versa. However, while PeaCE generates the code for composing blocks of the model both is SW and HW, it lacks code generation support for the blocks themselves and thus requires the definition of HW-SW blocks in later stage, which is time consuming when targeting several kinds of platforms.

Another interesting approach is SystemCoDesigner from the University of Erlangen-Nuremberg [8]. SystemCoDesigner is an actor-oriented approach using a high-level language named SysteMoC, built on the top of SystemC. It intends to generate HW-SW SoC implementations with automatic design space exploration techniques. The model is translated into behavioral SystemC model as a starting point for HW and SW synthesis. However, the HW synthesis is delegated to a commercial tool, namely, Forte's Cynthesizer, to generate RTL code from their SystemC intermediate model.

Several Simulink-based approaches have been also proposed to address HW-SW codesign [9]. Simulink, which was initially intensively used for simulation purposes, is becoming a good candidate for model-based design, particularly after the recent development of HW-SW synthesizers. Tools such as Real-Time Workshop for SW, or HW generators such as Synopsys Synphony HLS, Xilinx System Generator or Mathworks HDL coder are examples of these approaches. However, such methods are not always "off the shelf" and require the deep knowledge of a set of commercial tools and their appropriate usage.

Most of the approaches presented in the literature delegates the HW synthesis to commercial HLS tools. Mentor's Catapult, Synopsys's Synphony C Compiler, or Forte's Cynthesizer to name but a few are used with that purpose. Our approach shows such capabilities using free and open source tools. We believe that it is more flexible starting point, since those synthesizers can be easily tuned to target particular requirements.

3. Proposed Methodology

The paper presents a codesign environment that intends to address some of the limitations present in the state of the art particularly supporting SW and HW synthesis of the same source code with the synthesis of SW that scales on multicore platforms. It is expressly thought for the design of streaming signal processing systems. The essentials of the design flow are illustrated in Figure 1 and consist of the following stages.

(i) Dataflow-Based Modelling. We use an extension of the dataflow process network model (DPN), which is closely related to KPN, that enables to express a large class of applications, where processes, named actors, are written using a formal domain-specific language with the useful property of preserving a high degree of analyzability [10].

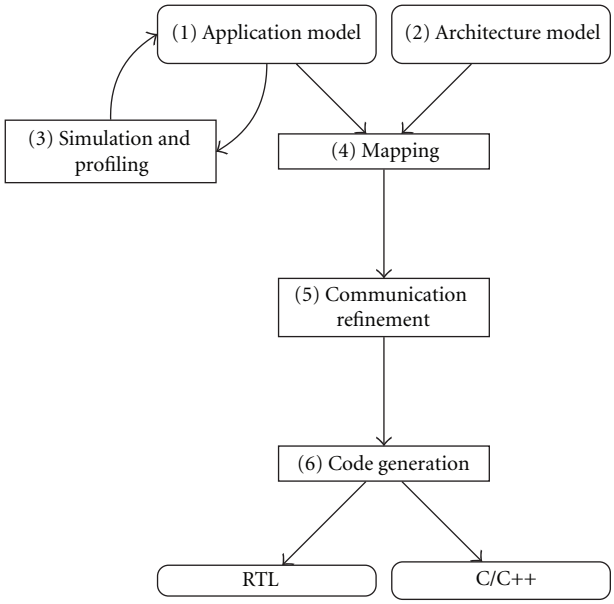


Figure 1: Overview of the design flow.

(ii) Architecture Model. We also employ an architecture model based on [5, 11] that enables to specify any architecture model for heterogeneous platforms composed by multicore processors and FPGAs at a high-level of abstraction.

(iii) Simulation and Profiling. We provide tools for functional validation, performance, and bandwidth estimations.

(iv) Mapping. HW-SW mapping can be both based on designer experience, or based on extracted metrics from the high level profiling or by more accurate profiling metrics if available from the platforms. Scheduling of SW partitions issues and available approaches are also discussed.

(v) Automatic Communication Interface Synthesis. Communication scheduling for interpartition communication as well as interfaces are automatically inserted in the design to be taken into account at synthesis stage.

(vi) Automatic Code Generation. HW and SW are automatically generated from CAL using ORCC and OpenForge synthesizers, including multicore support for SW components.

The codesign environment is implemented as an Eclipse plug-in built on the top of ORCC and OpenForge, open source tools that provide simulation and HW-SW synthesis capabilities. More details related to those capabilities are provided in the following sections. The complete tool chain is illustrated in Figure 2. The inputs of the tool chain are application (XDF) and architecture (IP-XACT) descriptions. The application is made by instantiating and connecting actors taken from an actor database (CAL).

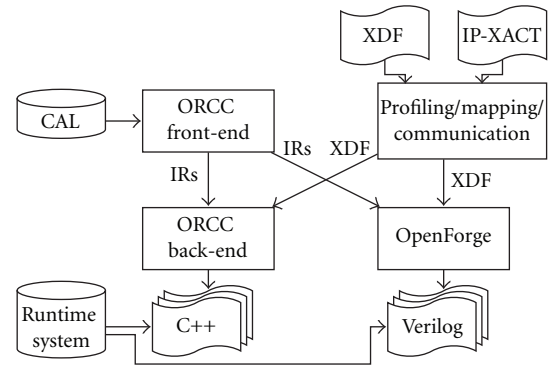


Figure 2: The cosynthesis tool chain.

4. Application Model: Dataflow with Firing

The dataflow paradigm for parallel computing has a long history from the early 1970s. Important milestones may be found in the works of Dennis [12] and Kahn [13]. A dataflow program is conceptually represented as a directed graph where vertices (named actors in the rest of the document) represent computational units, while edges represent streams of data. Figure 3 depicts a possible dataflow program. Formal dataflow models have been introduced in the literature, from Kahn process network (KPN) to synchronous dataflow (SDF) just to name a few. They differ by their models of computation (MoC) that define the behavior of the execution of the dataflow programs. There exists a variety of MoCs which results into different tradeoffs between expressiveness and efficiency.

In the paper, we use a model based on an extension of the dataflow process network MoC (DPN) [14]. Following the DPN MoC, actors execute by performing a number of discrete computational steps, also referred to as firings or firing functions. During a firing, an actor may consume data from input streams, produce data on output streams, and modify its internal state. An important guarantee is that internal states are completely encapsulated and cannot be shared with other actors, that is, actors communicate with each other exclusively through passing data along streams. This makes dataflow programs more robust and safe, regardless of the interleaving of actors. A firing rule is associated to each firing function, which corresponds to a particular pattern matching on the input streams and the current state. A firing occurs when its firing rule is satisfied, atomically and regardless of the status of all other actors.

In [10], authors presented a formal language for writing actors. The language, called CAL, is designed to express actors that belong to the DPN MoC. Of course, the language supports implementations of actors that can belong to more restricted MoCs, for example, CSDF, SDF, and so forth. The CAL language is a domain-specific language, it makes it possible to analyze actors easily and then determine their associated MoCs. It is an important property for many efficient optimizations that can be applied during code generation. An example of such optimizations is the static scheduling of actors (a correct-by-construction sequence of

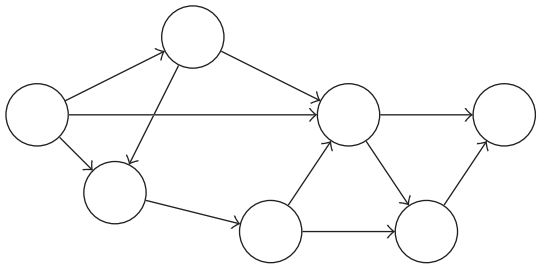


Figure 3: A simple dataflow program as a graph.

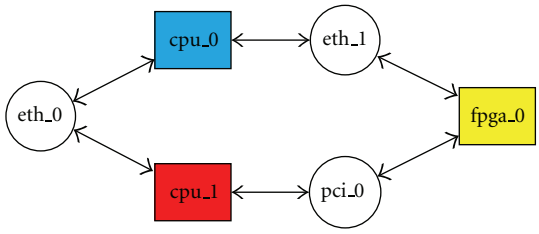


Figure 4: An example of a ring architecture composed of 3 operators and 3 media.

firings that can be executed without testing their firing rules) for some network partitions [15, 16].

CAL can also be directly synthesized to software and hardware [17–19]. Recently a subset of CAL, named RVC-CAL, has been standardized by ISO/IEC MPEG [20]. It is used as reference software language for the specification of MPEG video coding technology under the form of a library of components (actors) that are configured (instantiations and connections of actors) into networks to generate video decoders.

5. Architecture Model

The architecture model used in the design flow presented here, is based on the model proposed in [5, 11]. The architecture is modeled by an undirected graph where each vertex represents an operator (a processing element like a CPU or an FPGA in the terms of [11]) or a medium of communication (bus, memories, etc.), and edges represent interconnections viewed as a transfer of data from/to operators to/from media of communication. The model supports point-to-point or multipoint connections between operators.

The architecture model is serialized into an IP-XACT description, an XML format for the definition, and the description of electronic components, an IEEE standard originated from the SPIRIT Consortium. The architecture description is hierarchical and permits to describe architectures with different levels of granularity. For instance, a multicore processor can be represented as an atomic vertex or hierarchically exposing lower level details, where cores and memories become in turn atomic vertices. Figure 4 depicts a possible architecture with 3 operators connected with 3 point-to-point media.

6. Simulation and Profiling

RVC-CAL is supported by an interpreter that can simulate a dataflow graph of CAL actors. This interpreter is part of the Open RVC-CAL Compiler (ORCC). ORCC is a compiler infrastructure dedicated to RVC-CAL language [21]. More details on the ORCC infrastructure may be found in [18]. Essentially, the front end of ORCC transforms each actor into a corresponding intermediate representation (IR). The IR is then interpreted by the simulator.

A profiler, built on top of the interpreter, allows the user to extract high-level metrics when the execution of the dataflow program is simulated. The goal of the instrumentation is to determine the complexity of the actors. This complexity is extracted by counting instructions (assignment, load and store from/to the memory, loops, and if-statements) and operators in expressions (add, sub, mod, div, lsh, etc.). The instrumentation enables to extract metrics at an early stage of the system design without any information on the target architecture.

The relative bandwidth of FIFOs is also extracted in the profiling stage. The profiler can extract FIFO-related metrics by counting the number and size of data that are exchanged during execution.

7. Algorithm-Architecture Mapping

7.1. Partitioning. The partitioning consists of assigning each actor to a processing element (PE). A partition is defined as a subset of the application graph associated to each PE. In the proposed discussion of the design flow and associated results, partitions are statically defined (there is no actor migration from a partition to another at runtime). The static partitioning can be assigned manually according to the designer experience and/or requirements or automatically by using any optimization method aiming at optimizing appropriate objective function, using the metrics extracted during the profiling stage. However, design space exploration techniques that end up to automatic partitioning are not discussed in the rest of the paper. For more details on objective functions and associated heuristics, readers may refer to [22].

7.2. Scheduling. Once the partitioning is determined, the scheduling of actors assigned on a given PE consists of ordering their executions. In fact, the definition of appropriate scheduling strategies is not necessary in the context of reconfigurable hardware, since all the actors can run in parallel, regardless of the status of other actors. However, when the number of actors bound to a single PE is larger than one, a scheduler needs to be defined to execute actors sequentially. Defining a scheduling consists of determining an ordered list of actors for execution on a given partition. Such list can be constructed at compile time (static scheduling) or at runtime (dynamic scheduling). In our case, the scheduling of actors is deferred to runtime, since all of them are assumed to belong to the DPN MoC. In other words, the scheduler always needs to check if a given actor is enabled before execution, based on its firing rules. A simple scheduling strategy has

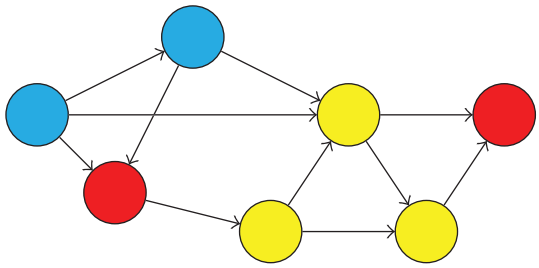


Figure 5: Mapping of the initial application graph.

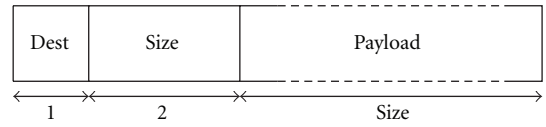


Figure 6: Header and the payload of the stored data in the serialization FIFO.

been selected that consists of using a round-robin scheduling algorithm, where each actor is checked one after the other by the scheduler for execution. Once an actor is selected, the scheduler tries to run it as long as it can while it matches one of its firing rules.

The runtime checks result in a significant runtime overhead [23] due to a large number of (sometimes unnecessary) conditional statements. However, scheduling statically (a subset of) those actors are sometimes possible when they belong to more restricted MoCs, namely, SDF and CSDF MoCs, that can help to reduce the overhead by removing some of those unnecessary checks. Several studies are devoted to solve this problem using other approaches and some interesting results showing relevant improvements are discussed in [23, 24].

The mapping stage is illustrated in Figure 5, where vertices of the dataflow graph are associated to PEs (the color of vertices comes from Figure 4).

8. Communication Scheduling

The HW-SW mapping results in a transformed dataflow graph. Transformations still need to be applied on the dataflow graph in order to exhibit the communications across partitions. The process mainly consists of transforming the initial dataflow graph by inserting additional vertices that represent communications between partitions, using the appropriate media between PEs from the architecture. Such transformation introduces special vertices in the application graph, which will encapsulate at a later stage the (de)serialization of data and the inclusion of the corresponding interfaces between partitions. This step is illustrated in Figure 1 where (de)serialization (resp., Ser. and Des.) and interface vertices are inserted.

The underlying DPN application model prevents from being able of scheduling communications statically. The serialization has the objective of scheduling the communications between actors that are allocated on different partitions at runtime. The fact is that when several edges from the dataflow graph are associated to a single medium of the architecture, data need to be interlaced in order to be able to share the same underlying medium.

In the case of serialization, on the sender side, "virtual" FIFOs are used to connect the serializer to incoming actors. By contrast with conventional FIFOs that store data and maintain the state (read/write counters), "virtual" FIFOs just maintain the state while data are directly stored into a single FIFO, shared by incoming actors. The idea behind such procedure is to emulate the history of FIFOs (emptiness, fullness) in order to fairly schedule data in the serialization FIFO. Data are scheduled by actors themselves, without using any scheduling strategy in the serializer. In order to retrieve data on the receiver side, a header is placed at the beginning of each data that defines the destination FIFO and the size (in byte) of the payload. This simple header is illustrated on Figure 6. On the receiver side, conventional FIFOs are used to connect the deserializer to outgoing actors. The deserializer is responsible to decode the header and put the payload to the appropriate destination FIFO.

For instance, in Figure 5, the blue partition has two outgoing FIFOs connected to the red partition. Thus, a serializer vertex is inserted in Figure 7.

Reconfigurable hardware and multicore processors can invoke various interprocess communication methods (shared memory, sockets, etc.) through various physical/logical interconnections (PCI Express, Ethernet, etc.) to implement the interaction between PEs. Interfaces communicate with other components via appropriate I/O. Interfaces are introduced during the synthesis stage and must be supported by libraries according to the nature of the PE presents on the platform. On the sender side, data from the serialization FIFO are packed (for instance, we use the maximum transmission unit in case of ethernet) and sent using the appropriate interface. On the receiver side, data are just unpacked and passed to the deserializer. In Figure 7 a PCIe interface is inserted and connected to the serializer previously cited.

9. Hardware and Software Code Generation

9.1. Software Synthesis. The software synthesis generates code for the different partitions mapped on SW PEs. ORCC compiler infrastructure is used to generate source code. The front end transforms each actor into a equivalent intermediate representation (IR). Then, a back end that translates the IR into C++ code has been developed. A naive implementation of a dataflow program would be to create one thread per actor of the application. However, in general, from the efficiency point of view it is not a good idea. In fact, distributing the application among too many threads results into too much overhead due to many context switches. A more appropriate solution that avoids too many threads is presented in [14]. It consists of executing actors in a single thread and using a user-defined scheduler, that selects the sequence of actor execution. The extension to multicore

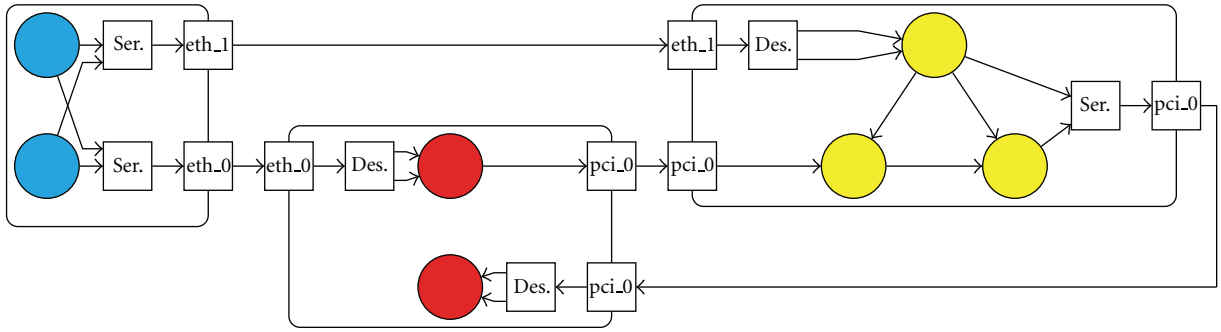


Figure 7: Communication refinement of the input application graph.

creates as many threads as existing cores. Since each core executes a single thread, threads are executed in parallel.

9.2. Hardware Synthesis. For the generation of the executable on programmable HW units, a synthesizable HDL code is automatically generated from the CAL dataflow program assigned to FPGAs. OpenForge, the HW synthesizer, is used to generate RTL code based on behavioral synthesis method translating the IR into HW descriptions expressed in Verilog [25] that can then be synthesized into implementations on FPGAs. More details about Openforge synthesis capabilities can be found in [17].

9.3. Runtime System. Runtime system libraries are needed for both SW and HW. Those libraries implement the FIFOs, serializers, deserializers, and the instantiation of the appropriate interfaces. On the software side, the dedicated runtime system provides the supports for the multicore requirements (threads, thread-safe queues, etc.). Note that the runtime is cross-platform and has been successfully tested on x86, PowerPC, and C66x DSP.

10. Experiments

10.1. Experiments on Multicore. CAL and its ISO standard subset RVC-CAL have been used with success for implementing video decoding algorithms in HW as reported and demonstrated in [17], and the dataflow model is clearly a good approach for exploiting massive parallel platforms such as FPGAs. Targeting less massively parallel systems, such as multicore platforms, it requires in addition appropriate methodologies for yielding efficient partitioning and scheduling.

The goal of the investigations presented here is to show how scalable parallelism can be achieved, in other terms that applications can be written at high level and their implementations can run faster when more parallelism is available in the implementation platform.

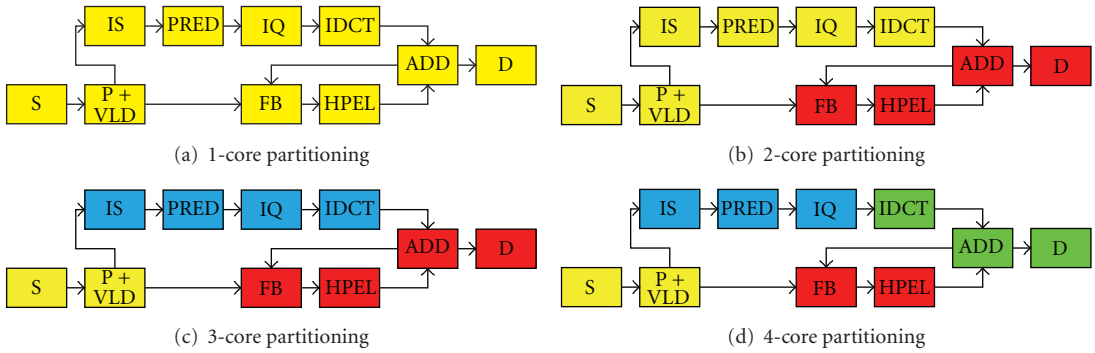
Prior researches have already reported implementation results of CAL programs on multicore. In [19], an implementation of an MPEG-4 SP decoder running on 2-core processor is reported. The ACTORS project [26] has reported the implementation of an MPEG-4 SP decoder with rather good speedup on 4-core processors [27]. The experimental results reported here show the evolution of these works in terms of improved scalability, portability on different platforms and increased throughput for the same application example. In this case study we have implemented 2 versions of an MPEG-4 Simple Profile decoder onto 2 different multicore platforms.

(i) Serial MPEG-4 Simple Profile. It is illustrated in Figure 8. It contains 13 actors that correspond to the entropy decoding (syntax parser and the variable length decoder), the residual decoding (AC-DC predictions, inverse scan, inverse quantization, and IDCT), and the motion compensation (framebuffer, interpolation, and residual error addition). The source (S) reads the bitstream, while the sink (D) displays the decoded video.

(ii) YUV-Parallel MPEG-4 Simple Profile. It is illustrated in Figure 9. The so-called parallelism is due to the fact that the color space components Y, U, and V can be decoded separately. It is composed by 33 instances of actors. Each branch includes two subnetworks "TX" and "MX", where "X" is to be replaced by the appropriate component, that, respectively, corresponds to the residual decoding and the motion compensation. The "P" subnetwork (with 7 actors) corresponds to the entropy decoder and finally the "M" actor merges the decoded components and sends data to the display (D).

Two implementation platforms have been used: a desktop computer with an Intel i7-870 processor, with 4 cores at 2.93 GHz, and a Freescale P4080 platform, using a PowerPC e500 processor with 8 cores at 1.2 GHz. The result of the SW synthesis from the dataflow program is a single executable file that is configured using the parameters that defines the partitions. A list of actors is extracted from each partition. Each list of actors is then used to create a thread where actors are instantiated and scheduled. Note that in both cases, it is a single executable file that is running on 1, 2, 3, or 4 cores. Three sequences at different resolutions have been encoded at different bitrates: foreman (QCIF, 300 frames, 200 kbps), crew (4CIF, 300 frames, 1 Mbps), and stockholm (720p60, 604 frames, 20 Mbps).

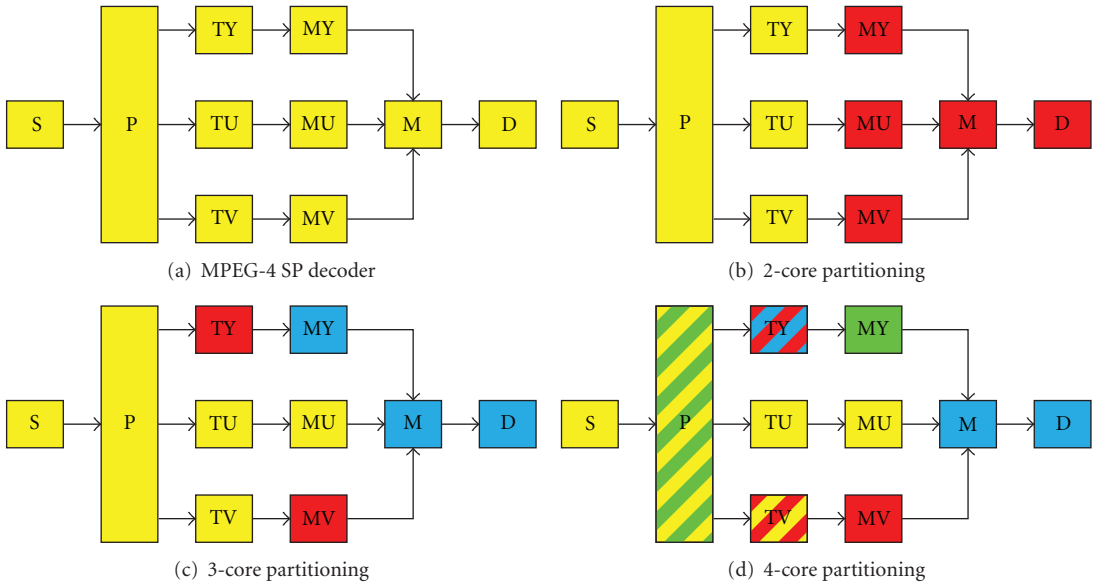
Figures 8 and 9 describe the different partitions used on the serial and the parallel versions of the decoders



(a) 1-core partitioning (b) 2-core partitioning

(c) 3-core partitioning (d) 4-core partitioning

Figure 8: Partitions of the serial version of the MPEG-4 SP decoder for 1, 2, 3, and 4 cores configurations.



(a) MPEG-4 SP decoder (b) 2-core partitioning

(c) 3-core partitioning (d) 4-core partitioning

Figure 9: Partitions of the parallel version of the MPEG-4 SP decoder for the 1, 2, 3, and 4 cores configurations.

Table 1: Framerate of the serial MPEG-4 SP decoder at QCIF, SD and HD resolutions.

Platform	Resolution	Framerate (no. of cores)			
		1	2	3	4
Intel i7-870	176 × 144	1788	3426	4416	5260
	704 × 576	126	203	219	307
	1280 × 720	37	63	71	84
Freescale P4080	176 × 144	275	520	676	913
	704 × 576	18	30	43	51
	1280 × 720	6	10	12	16

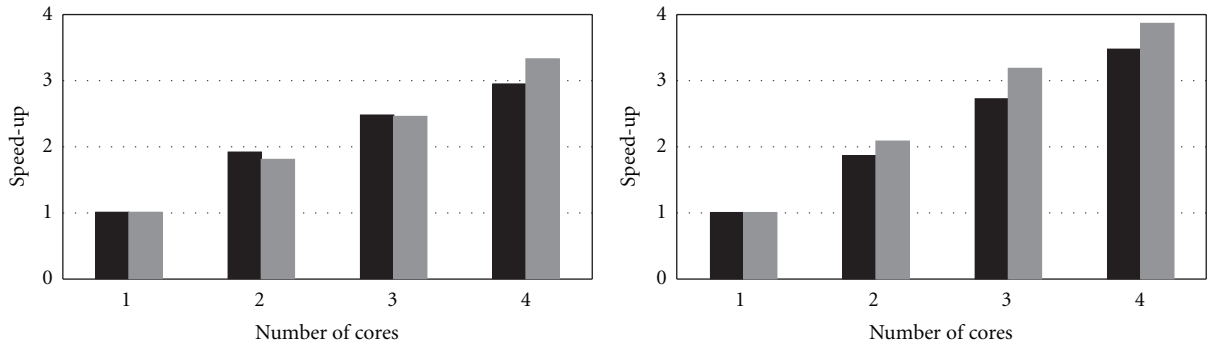
Table 2: Framerate of the YUV-parallel MPEG-4 SP decoder at QCIF, SD and HD resolutions.

Platform	Resolution	Framerate (no. of cores)			
		1	2	3	4
Intel i7-870	176 × 144	1580	2940	4303	5494
	704 × 576	104	178	267	340
	1280 × 720	34	62	75	89
Freescale P4080	176 × 144	223	465	711	853
	1280 × 720	15	30	43	52
	1280 × 720	5	9	13	18

respectively. The blocks represented by a stripe background are distributed over different partitions.

Tables 1 and 2 report the framerate in frame per second (fps) of the serial and the parallel decoders, respectively, on the 4 cores.

The resulting speedup is illustrated in Figure 10. It shows that it is possible to achieve significant speedups when adding more cores. It is also observed in the experiment that a near to linear speedup in most of the cases is obtained, which tends to indicate that the dataflow applications scale well on multicore processors. Particularly, on the P4080 at QCIF resolution, it is observed a more than linear scale. This anomaly is due to the reduction of the scheduling overhead when the number of the partitions increases.



(a) Speedup for the serial decoder at QCIF resolution

(b) Speedup for the parallel decoder at QCIF resolution

Figure 10: Speedup of the MPEG-4 SP decoder running on an X86 quad-core (in black) and on a PowerPC e500 8-core (in gray) processors.

Table 3: RVC-CAL JPEG encoder throughput for one image in ms.

Type	Resolution	FPGA Frequency	
		50 Mhz	80 Mhz
RVC-CAL HDL	512×512	48 ms	28.9 ms
Generated Code	1920×1080	373 ms	223 ms
Handwritten VHDL	512×512	31.2 ms	18.7 ms
	1920×1080	317 ms	190 ms

Figure 11 reports results related to the scalability that is also preserved when changing the resolution of the video format. The results show that both decoders produce real-time performances for HD resolution (720p60) starting from the 2-core configurations. Results that are remarkable considering the high-level starting point and the lack of any low-level and platform-specific optimizations. In terms of speedup factor versus the single-core performance, the results are slightly better than the ones presented in [27].

In term of absolute throughput, the experiment shows a very significant improvement. The normalized throughput results in macroblock (A macroblock corresponds to 16×16 pixels in MPEG-4 SP, which is equivalent to $6 \times 8 \times 8$ bytes in 4 : 2 : 0 format.) per second divided by the frequence—$\text{MB} \cdot \text{s}^{-1} \cdot \text{Hz}^{-1}$ are 5.94×10^{-6} $\text{MB} \cdot \text{s}^{-1} \cdot \text{Hz}^{-1}$ for the ARM11 in [27], 22.68×10^{-6} for the PowerPC and 60.41×10^{-6} for the Intel i7-870 (for the serial decoder at QCIF resolution).

Figure 11 shows that the parallel version of the decoder scales much better than the serial one. Moreover, the higher the resolution, the lower the speedup factor we obtain. This result is due to the fact that the application is constituted by a lower number of actors, a fact that reduces the number of possible partitions, thus reducing the possibility to balance equitably the processing load. By contrast, the parallel version is less sensitive to the resolution. Those results indicate that the parallel decoder seems to be a better starting point when targeting implementations on more cores.

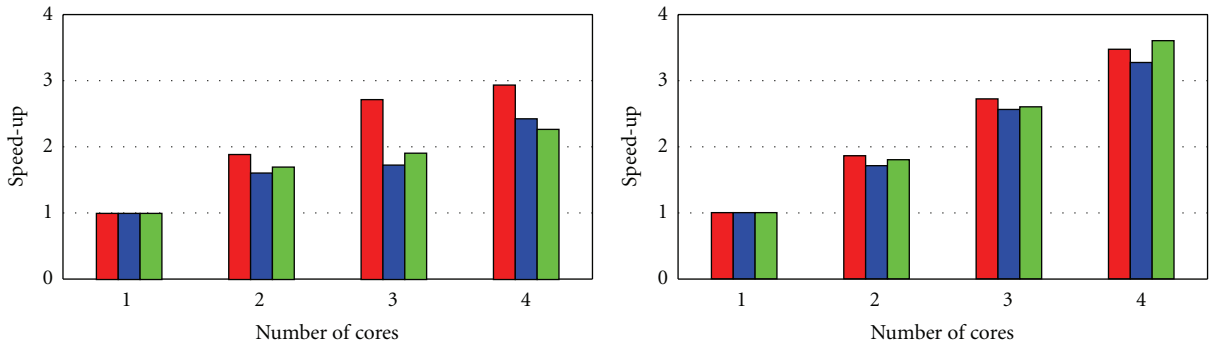
10.2. Experiments on Reconfigurable Hardware. The purpose of this experiment is to compare the HDL synthesis from a dataflow application with a handwritten one. To this end, a baseline profile JPEG encoder was developed and a VHDL JPEG encoder was taken from the OpenCores project [28]. Figure 12(a) represents the dataflow JPEG encoder where computation blocks are at the encoding DCT, quantization (Q), zigzag scan (ZZ), and variable-length encoding (VLC). As for the VHDL encoder which is represented in Figure 12(b) the DCT, Q, and ZZ are processed in parallel for the luma (Y) and chroma (UV) blocks. This VHDL encoder design was chosen for its dataflow resemblance with the RVC-CAL JPEG encoder.

The ML509 Virtex 5 board was used for both encoders. Table 3 indicates the throughput of both encoders for encoding two images with different resolutions and the throughput of this images for two different clock frequencies. The result of this experiment shows that the handwritten VHDL JPEG encoder is only 1.5 times faster when compared to the automatically generated HDL from the initial version of the dataflow JPEG encoder. One of the reasons why the VHDL encoder is faster is that it uses the FDCT IP core accelerator from Xilinx and that the luma and chroma blocks are processed in parallel. The splitting of the Y, U, and V components is a possible optimization for the RVC-CAL JPEG encoder.

Table 4 compares the resource usage of both encoders on the FPGA. The generated JPEG encoder uses 5% less registers, 4 times less LUT-FF pairs, and 2% more of DSP48E1 (which represents only 3% of the DSP block on a Virtex 5 FPGA, those DSP blocks are mainly used in the FDCT and in the quantization actors) than the handwritten VHDL JPEG encoder. Thus the generated HDL from the high-level version of the RVC-CAL JPEG encoder requires less FPGA logic resources than the handwritten one.

We can also notice that the RVC-CAL JPEG encoder can encode 4 Full HD images (1920×1080) in less than a second at 80 MHz and it can encode in real-time 512×512 images.

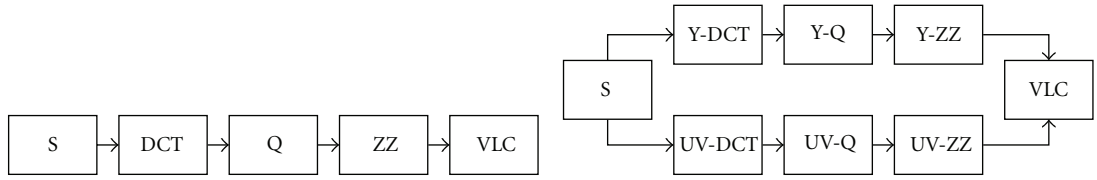
10.3. Synthesis of Both HW and SW from the Same Dataflow Description. The goal of the experiment is to validate the portability for the proposed design approach. To this end, additional to the JPEG encoder, we developed a JPEG decoder in RVC-CAL for creating a baseline profile JPEG



(a) Speedup for the serial decoder according to the resolution

(b) Speedup for the parallel decoder according to the resolution

Figure 11: Speedup of the MPEG-4 SP decoder on an Intel i7-870 processor at QCIF (red), SD (blue) and HD (green) resolutions.



(a) RVC-CAL JPEG Encoder

(b) Handwritten VHDL JPEG Encoder

Figure 12: Description of the RVC-CAL and handwritten VHDL JPEG encoders.

codec. We have implemented this codec onto three different platforms made of FPGAs and embedded processors. Figure 13 represents the JPEG codec. The host used in all platforms is the one that was used in the first experiment. The two first platforms are FPGA-based platforms. The first one is a proprietary board with a Xilinx Spartan3 FPGA and the second one is a ML509 board with a Virtex-5 FPGA as used in the previous experiment. The last platform is composed by a Freescale P2020 board, with 2 cores at 1 GHz. Ethernet, PCI express (PCIe) and RS232 are used to communicate between the host and the specialized PEs.

It has to be noticed that several mappings have been successfully tested. In fact, a single actor can seamlessly be synthesized to general-purpose processors, embedded processors, and FPGAs. A partition of the dataflow application can be swapped from SW to HW and vice versa, and all yield functionally equivalent implementations. For the sake of clarity, results are given only for a meaningful partitioning, separating the encoding and the decoding processes. More precisely, the partitioning of the application consists of assigning the whole encoding process on the specialized PE and the decoding process is done by the host. Note that on the P2020 the encoding is balanced between the 2 cores.

Results of the experiment are summarized in Table 5. Three different media of communication have been tested ("—" indicates that the corresponding communication link has not been tested in the executed experiments). The results using the serial link present low performances in term of fps. At least, it makes explicit that several interfaces can be used in the design flow.

The results using the Virtex5 and the PCI express interface are competitive with the ones presented in [6, 8].

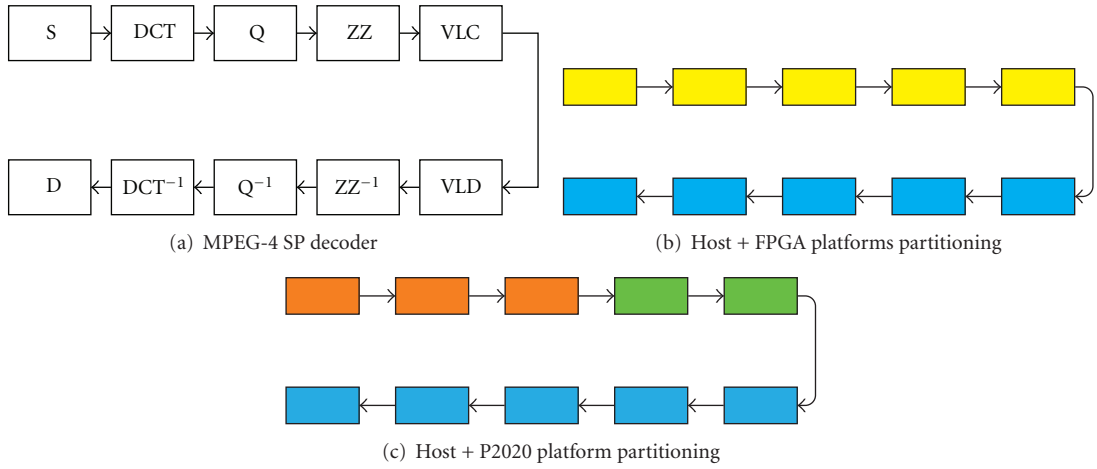
Table 4: FPGA occupation of the handwritten VHDL JPEG encoder versus the RVC-CAL JPEG Encoder.

Logic utilization	FPGA occupation			
	Handwritten		Generated	
	Usage	%	Usage	%
Registers	17869	11	10965	6
Slice LUTs	16439	19	14413	18
LUT-FF Pairs	11817	64	3504	16
Block RAM	35	13	43	14
DSP48E1s	2	1	18	3

Table 5: Framerate of the JPEG codec on 3 platforms with 3 interfaces and a 512×512 video resolution.

	Serial link	Ethernet	PCIe
Spartan3	0.2	3.7	N.A.
ML509 with Virtex5	0.2	—	8.5
P2020 with PowerPC	—	3.8	—

On the one hand, the encoder only implemented on the Virtex-5 can encode around 4 full HD frames per second at 80 MHz. While on the other hand, the decoder only, on the host, is able to decode at 135 fps 512×512 frames. This result indicates that either the interface bandwidth or the communication scheduling is the limit for the design performance.



(a) MPEG-4 SP decoder

(b) Host + FPGA platforms partitioning

(c) Host + P2020 platform partitioning

Figure 13: Description of the JPEG codec and the partitioning for the platforms.

11. Conclusions

The presented approach provides a unified design flow for the automatic synthesis of SW and HW components. The synthesis is directly obtained from high-level descriptions of both application programs and platform architectures. The high degree of abstraction enables the program to undergo several design iterations that enable rapid prototyping of applications onto architectures, validation, and testing of performances for different mappings by relying on automatic cosynthesis tools. Indeed, it consumes much less resource to refactor the dataflow program. In general, it is not possible to map imperative programs arbitrarily onto platforms without code rewriting. Rewriting the code, to fit a given platform, is time consuming and error prone, and usually most of the design time is wasted in the phase of debugging to reobtain design that works correctly. In our design flow, the design can go through many more design iterations, with less effort since just the mapping needs to be changed, that can shorten the path to implementation.

The paper has demonstrated, by experiments, both the scalability and the portability of real-world applications. Several implementations have been validated onto different platforms composed of both FPGAs and multicore processors. The results reported in the paper have only addressed and demonstrated the portability and scalability features provided by the approach. However, the potential of the approach can progress much further in terms of design space exploration capabilities. Future investigations will focus on the development of tools for automatic design space exploration, driven by objective functions that will use the metrics extracted during the profiling stage.

References

[1] G. De Micheli, "Hardware synthesis from C/C++ models," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, pp. 382–383, 1999.

[2] R. K. Gupta and G. De Micheli, "Hardware-software cosynthesis for digital systems," *IEEE Design and Test of Computers*, vol. 10, no. 3, pp. 29–41, 1993.

[3] A. Kalavade and E. A. Lee, "A hardware-software codesign methodology for dsp applications," *IEEE Design and Test of Computers*, vol. 10, no. 3, pp. 16–28, 1993.

[4] F. Balarin, M. Chiodo, P. Giusto et al., *Hardware-Software Co-Design of Embedded Systems: The POLIS Approach*, Kluwer Academic Publishers, Norwell, Mass, USA, 1997.

[5] T. Grandpierre, C. Lavarenne, and Y. Sorel, "Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors," in *Proceedings of the 7th International Conference on Hardware/Software Codesign (CODES'99)*, pp. 74–78, Rome, Italy, May 1999.

[6] T. Stefanov, C. Zissulescu, A. Turjan, B. Kienhuis, and E. Deprette, "System design using Khan process networks: the Compaan/Laura approach," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, vol. 1, pp. 340–345, February 2004.

[7] S. Ha, S. Kim, C. Lee, Y. Yi, S. Kwon, and Y. P. Joo, "PeaCE: a hardware-software codesign environment for multimedia embedded systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 12, no. 3, Article ID 1255461, 2007.

[8] J. Keinert, M. Streubuhr, T. Schlichter et al., "SystemCoDesigneran automatic ESL synthesis approach by design space exploration and behavioral synthesis for streaming applications," *ACM Transactions on Design Automation of Electronic Systems*, vol. 14, no. 1, article no. 1, 2009.

[9] S. A. Butt, P. Sayyah, and L. Lavagno, "Model-based hardware/software synthesis for wireless sensor network applications," in *Proceedings of the Electronics, Communications and Photonics Conference (SIECPC '11)*, pp. 1–6, April 2011.

[10] J. Eker and J. Janneck, "CAL Language Report," Tech. Rep. ERL Technical Memo UCB/ERL M03/48, University of California at Berkeley, December 2003.

[11] M. Pelcat, J.F. Nezan, J. Piat, J. Croizer, and S. Aridhi, "A system-level architecture model for rapid prototyping of heterogeneous multicore embedded systems," in *Proceedings of the Conference on Design and Architectures for Signal and Image Processing (DASIP '09)*, 2009.

[12] J. B. Dennis, "First version of a dataflow procedure language," in *Proceedings of the Symposium on Programming*, pp. 362–376, 1974.

[13] G. Kahn, "The semantics of simple language for parallel programming," in *Proceedings of the IFIP Congress*, pp. 471–475, 1974.

[14] E. A. Lee and T. M. Parks, "Dataflow process networks," *Proceedings of the IEEE*, vol. 83, no. 5, pp. 773–801, 1995.
[15] J. Eker and J. W. Janneck, "A structured description of dataflow actors and its applications," Tech. Rep. UCB/ERL M03/13, EECS Department, University of California, Berkeley, 2003.
[16] M. Wipliez and M. Raulet, "Classification and transformation of dynamic dataflow programs," in *Proceedings of the Conference on Design and Architectures for Signal and Image Processing (DASIP '10)*, pp. 303–310, 2010.
[17] 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," *Journal of Signal Processing Systems*, vol. 63, no. 2, pp. 241–249, 2011.
[18] M. Wipliez, G. Roquier, and J. F. Nezan, "Software code generation for the RVC-CAL language," *Journal of Signal Processing Systems*, vol. 63, no. 2, pp. 203–213, 2009.
[19] I. Amer, C. Lucarz, G. Roquier et al., "Reconfigurable video coding on multicore: an overview of its main objectives," *IEEE Signal Processing Magazine*, vol. 26, no. 6, pp. 113–123, 2009.
[20] ISO/IEC 23001-4:2009, "Information technology—MPEG systems technologies—Part 4: Codec configuration representation," 2009.
[21] "Open RVC-CAL Compiler," http://orcc.sourceforge.net/.
[22] M. Mattavelli, C. Lucarz, and J.W. Janneck, "Optimization of portable parallel signal processing applications by design space exploration of dataflow programs," in *Proceedings of the IEEE Workshop on Signal Processing Systems*, 2011.
[23] J. Ersfolk, G. Roquier, F. Jokhio, J. Lilius, and M. Mattavelli, "Scheduling of dynamic dataflow programs with model checking," in *Proceedings of the IEEE Workshop on Signal Processing Systems*, 2011.
[24] R. Gu, J. W. Janneck, M. Raulet, and S. S. Bhattacharyya, "Exploiting statically schedulable regions in dataflow programs," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '09)*, pp. 565–568, IEEE Computer Society, Washington, DC, USA, 2009.
[25] "OpenForge," https://sourceforge.net/projects/openforge/.
[26] "ACTORS Project," http://www.actors-project.eu/.
[27] A. Carlsson, J. Eker, T. Olsson, and C. von Platen, "Scalable parallelism using dataflow programming," in *Ericson Review*, On-Line Publishing, 2011, http://www.ericsson.com/.
[28] "OpenCores," http://opencores.org/.