Multiprocessor scheduling of dataflow models within the Reconfigurable Video Coding framework

Jani Boutellier and Victor Martin Gomez and Olli Silvén
Machine Vision Group
University of Oulu, Finland
{jani.boutellier, victor.martin, olli.silven}@ee.oulu.fi

Christophe Lucarz and Marco Mattavelli
Microelectronic Systems Laboratory
École Polytechnique Fédérale de Lausanne, Switzerland
{christophe.lucarz, marco.mattavelli}@epfl.ch

Abstract

The new Reconfigurable Video Coding (RVC) standard of MPEG marks a transition in the way video coding algorithms are specified. Imperative and monolithic reference software is replaced by a collection of interconnected, concurrent functional units (FUs) that are specified with the actor-oriented CAL language. Different connections between the FUs lead to different decoders: all previous standards (MPEG-2 MP, MPEG-4 SP, AVC, SVC) can be built with RVC FUs.

The RVC standard does not specify a schedule or scheduling heuristic for running the decoder implementations consisting of FUs. Previous work has shown a way to produce efficient quasi-static schedules for CAL actor networks. This paper discusses the mapping of RVC FUs to multiprocessor systems, utilizing quasi-static scheduling. A design space exploration tool has been developed, that maps the FUs to a multiprocessor system in order to maximize the decoder throughput. Depending on the interprocessor communication cost, the tool points out different mappings of FUs to processing elements.

1. Introduction

The effort of designing the Reconfigurable Video Coding (RVC) standard is motivated by the intent to describe already existing video coding standards with a set of common atomic building blocks (*e.g.*, IDCT). Under RVC, existing video coding standards are described as specific configurations of these atomic blocks, also knows as functional units (FUs). This greatly simplifies the task of designing future multi-standard video decoding applications and devices by allowing software and hardware reuse across standards.

The functional units are described in RVC with a dataflow/actor object-oriented language named CAL that allows concise description of signal processing algorithms. For this reason, CAL has been chosen as the language for the reference software of the standard. The CAL Model of Computation (MoC) describes decoders as a set of atomic blocks in a way that exposes parallelism between the computations. However, the abstract and high-level CAL models require a systematic implementation methodology and tools to implement these CAL models into real systems. One of the implementation problems is the assignment of RVC FUs to the processing elements (PEs) available in the underlying system, as well as generating efficient schedules for the FU actions.

In previous work [3], a methodology has been designed for transforming RVC CAL networks into a set of homogeneous synchronous dataflow (HSDF) [8] graphs that enable efficient quasi-static scheduling. The work presented in this paper takes as an input the set of HSDF graphs produced by the previous work, and tries to find an optimal mapping of RVC FUs to the PEs in the system. This paper describes a design space exploration (DSE) tool and as an example, shows the mapping of the RVC MPEG-4 Simple Profile (SP) decoder to a multiprocessor system. The number of processors is theoretically not limited, but interprocessor communication costs naturally lead to solutions that only have a couple of processing elements. Finally, the results provided by the DSE tool are discussed.

The paper is organized as follows. Section 2 explains the main concepts in the Reconfigurable Video Coding framework. Section 3 explains the used scheduling approach.

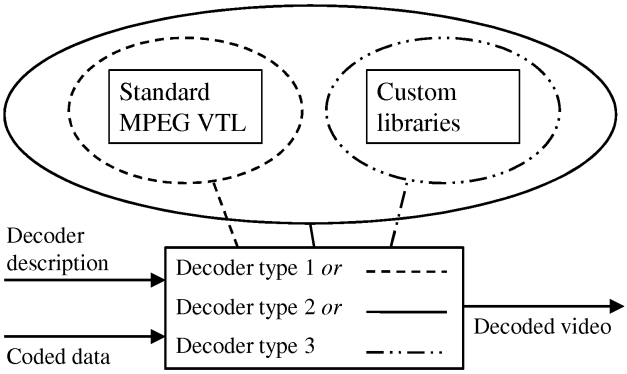

Figure 1. The RVC decoder can be instantiated from standard or custom FUs.

Section 4 illustrates the methodology on a real-life application (MPEG-4 Simple Profile decoder). Section 5 concludes the paper.

2. Concepts of the Reconfigurable Video Coding framework

The MPEG RVC framework aims to offer a more flexible and fast path to innovation of future video coding standards. The RVC framework also provides a high level specification formalism that establishes a starting point model for direct software and hardware synthesis. Moreover, the RVC framework intends to overcome the lack of interoperability between various video codecs that are deployed into the market. Unlike previous standards, RVC does not itself define a new codec. Instead, it provides a framework to allow content providers to define a multitude of different codecs, by combining together FUs from the Video Tool Library (VTL). Such a possibility clearly simplifies the task of designing future multi-standard video decoding applications and devices by allowing software and hardware reuse across video standards.

The main strength of RVC is that unlike the traditional video coding standards, where decoders used to be rigidly specified, a description of the decoder is associated to the encoded data, enabling a reconfiguration and instantiation of the appropriate decoder at the video data receiver. Figure 1 illustrates how the decoders can be constructed within the RVC framework. The MPEG RVC framework defines two standards: MPEG-B, which defines the overall framework as well as the standard languages that are used to describe different components of the framework, and MPEG-C, which defines the library of video coding tools employed in existing MPEG standards [10].

MPEG VTL is normatively specified using RVC-CAL. An appropriate level of granularity for blocks within the standard library is important, to enable efficient reuse within the RVC framework. If the library is too coarse, modules will be too large to allow reuse between different codecs. On the other hand, if the granularity is too fine, the number of modules in the library will be too large for an efficient and practical reconfiguration process, and may obscure the desired high-level description and modeling of the RVC decoder. Prior to RVC, reuse of components across applications has been done, *e.g.*, in multi-mode systems [11].

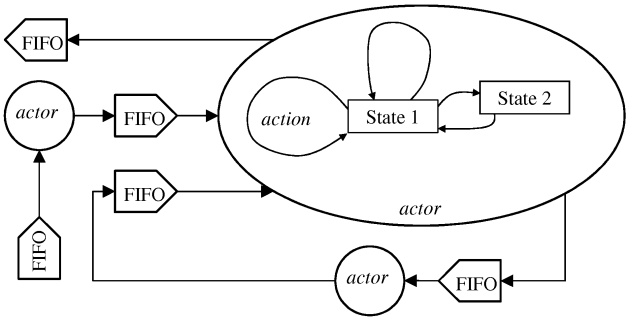

Figure 2. An arbitrary CAL actor network.

2.1. The CAL model of computation

CAL is a dataflow and actor oriented language that has been recently specified as a subproject of the Ptolemy project [5] at the University of California, Berkeley. The final CAL language specification has been released in December 2003 [4]. CAL models different algorithms by using a set of interconnected dataflow components called actors (see Figure 2).

An actor is a modular component that encapsulates its own state. The state of any actor is not sharable with other actors. Thus, an actor cannot modify the state of another actor. Interactions between actors are only allowed through input and output ports. The behavior of an actor is defined in terms of a set of actions. The operations an action can perform are to 1) consume input tokens, to 2) modify internal state and to 3) produce output tokens. The actors are connected to each other through FIFO channels and the connection network is specified with the Network Language (NL). The action executions within one actor are purely sequential, whereas at the network level, the actors can work concurrently. CAL allows also hierarchical system design: each actor can contain a network of actors.

A CAL actor can also be interpreted as an Extended Finite State Machine (EFSM), and the actions as EFSM state transitions (see Figure 3). The state-space of an EFSM is much greater than that of a regular FSM, because EFSMs (CAL actors) may contain variables. The state transitions in the CAL actors can not take place freely: four types of

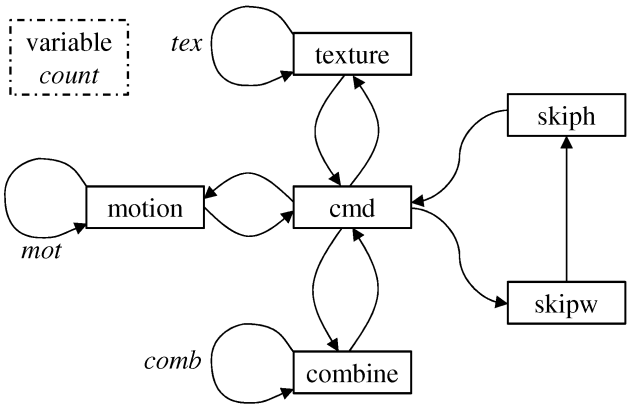

Figure 3. The CAL actor *add* interpreted as an extended finite state machine.

control mechanisms [9] define which action is going to execute next. 1) Availability of input tokens, 2) value of input tokens, 3) actor state and 4) action priorities. These control mechanisms increase the expressiveness of the CAL language, but unfotunately produce an overhead at run-time: actors are constantly checking the status of control mechanisms. The quasi-static scheduling approach used in this work [3] minimizes this run-time overhead by analyzing the CAL network at design-time, leaving only the necessary control mechanisms active at run-time.

3. The scheduling approach

Nowadays, a significant amount of video decoding takes place on mobile devices that have strict power and performance constraints. Thus, also the scheduling used in mobile video decoders must be done efficiently.

In quasi-static scheduling [7], most of the scheduling effort is done off-line and only some infrequent data-dependent scheduling decisions are left to run-time. The off-line determined schedule parts are collected to a repository that is used by the run-time system, which selects entries from the repository on demand and appends them to the ongoing program execution. This approach limits the number of run-time scheduling decisions and improves the efficiency of the system.

Quasi-static scheduling fits very well to the context of video decoding. The video decoding process of hybrid block-based decoders (such as MPEG-4 SP) consists of the decoding of *macroblocks* that consist of several *blocks*. For example, in MPEG-4 SP the blocks are of size 8x8 pixels, and six blocks form a macroblock that produces the information that can be seen on the screen in a 16x16 pixel area. The decoding process varies block-by-block, so the static schedule pieces in quasi-static scheduling should be of a

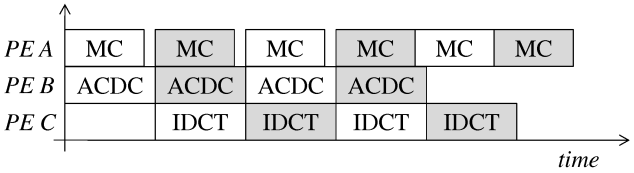

Figure 4. A quasi-static 3-PE macroblock decoding schedule consisting of 6 parts.

granularity of one block. Furthermore, quasi-static scheduling assumes that the scheduled tasks have been pre-assigned to processing elements at design time, which also reduces the run-time overhead of scheduling.

Figure 4 sketches a quasi-static schedule as described above. The decoding of even-numbered blocks is depicted with white tasks in the Gantt chart, whereas the odd-numbered blocks are gray. Blocks 4 and 5 only require Motion Compensation (MC) for decoding, whereas blocks 0 through 3 require also AC/DC prediction and IDCT. The figure simplifies the real-world computations: in reality each block (MC, AC/DC, IDCT) would consist of hundreds of CAL actions. The figure is simplified so far that it only discriminates the tasks on different PEs and different schedule parts. The detailed action schedules that are not shown, are computed at design time and stored for run-time use. The run-time system then selects the appropriate schedule part for decoding each block.

In our previous work [3], we have explained a procedure to transform RVC CAL networks into homogeneous static synchronous data flow (HSDF) graphs that can be quasi-statically scheduled. The quasi-static scheduling algorithm takes the CAL actors and their interconnecting networks as an input, and produces a set of HSDF graphs as an output. The number of produced HSDF graphs depends on the number of *modes* that the CAL network has; the different decoder modes represent the various decoding approches of 8x8 pixel blocks.

In each produced HSDF graph, one HSDF actor represents an instance of a CAL action. For example, if the RVC add (see Figure 3) actor executes the tex action 64 times, there will be 64 HSDF actors representing that action in the HSDF graph. Note that if a CAL actor is active in several different network modes M, the same HSDF actors will appear in each graph that represents those modes M.

The dataflow simulation environment used to run RVC on workstations, is named OpenDF [2]. We have modified the OpenDF simulator to enable quasi-static scheduling of the CAL actors, a work that is still somewhat in progress. Simultaneously, we are working with a *C* code version of RVC MPEG-4 SP produced by ORCC [6], to enable experimenting with multiprocessor schedules.

Table 1. Activity of the MPEG-4 SP actors in different operation modes.

FU	New frame	Inter block	ZMV block	Intra block	Hybrid block
Address	x	x	x	x	x
Buffer		x	x	x	x
Interpol.		x	x		x
DCSplit				x	x
DCRec.	x	x	x	x	x
IS	x	x	x	x	x
IAP	x	x	x	x	x
IQ				x	x
IDCT2D				x	x
Add	x	x	x	x	x

Table 2. Number of HSDF actors in the five mode graphs.

New frame	Inter block	ZMV block	Intra block	Hybrid block
6	444	442	345	592

4. Case study: MPEG-4 SP decoder

The behaviour of the MPEG-4 SP decoder (see Figure 5) is controlled to a great extent with the *btype* signal that is created in the *parser* actor and affects the behaviour of the *texture* and *motion* networks that do the main decoding effort. For our work, the *btype* signal was analyzed and it was discovered that it defines five major operations modes for the *texture* and *motion* networks. The combined *motion* and *texture* networks are depicted in Figure 6.

With the information about the different *btype* modes, the MPEG-4 SP decoder was profiled extensively to get the number of CAL actor action executions for each *btype* mode. This profiling produced as a result a list of actor activities that is depicted in Table 1. The table simplifies the profiling results in the way that any activity in the actor respective to the mode is marked with an x, independent of the number of actions executed. The total number of actions executed in each mode is described in Table 2.

4.1. Design space exploration

The focus of this work is to explore the mapping of the MPEG-4 SP actors to a multiprocessor system. The number of mapping alternatives is considerable and requires an automated approach.

The target architecture consists of homogeneous PEs that are fully connected. We define the task as a design space exploration (DSE) problem that has three parameters: a) mapping of each FU to one of the processing elements and b) determining the priorities between FUs and c) setting the cost of inter-processor communication (IPC). Those actors that belong to a higher-priority FU, are fired before the actors that belong to lower-priority FUs. Each FU is constrained to run completely on one PE, but one PE can be responsible for any number of FUs. For these experiments, the number of PEs was restricted to four to avoid the explosion of the number of possible solutions. However, when the IPC cost increases, number of PEs is naturally limited below four. The set of PEs is assumed to be fully interconnected.

Pino *et al.* [12] have considered a related problem of *clustering* SDF graphs on multiprocessors. In a clustering problem, an arbitrary graph is given, and the vertices (actors) must be grouped as clusters that are then assigned to processors. The clustering problem is essentially about discovering the sets of vertices that should belong together to one cluster. In our mapping problem the clustering step can be omitted, because the original CAL model defines the clusters: the HSDF actors belonging to one RVC FU form one cluster.

Our design-space exploration software takes as an input the set of HSDF graphs produced by our previous work [3], the number of clock cycles consumed by each CAL actor action, and the IPC cost, which is a simple integer constant. The DSE software searches the combination of FU $\Rightarrow$ PE mappings and FU priorities that produces the minimal combined makespan for all HSDF graphs. The combination of makespans is computed by summing together the makespans produced by scheduling each graph separately.

The priorities between FUs in the design-time scheduling slightly affects the resulting schedule makespans. An optimal solution would be to assign a different priority to each HSDF actor, but that would make the search space unfeasible. Thus, the priorities are assigned to the complete clusters (FUs).

The CAL actions in RVC are generally very fine grained and one action usually only modifies the value of a single pixel. We used the *C* language version of MPEG-4 SP to measure the latency of each action on an Altera Cyclone III FPGA running the Nios II/s soft processor [1]. The program was executed from an external DDR SDRAM and the processor had 512 bytes of instruction cache. Table 3 shows the average measured latencies for each action. However, the quasi-static scheduling approach used here requires the use of worst-case execution times.

The DSE software works in three phases. In the first phase, the software produces all FU $\Rightarrow$ PE mapping combinations and computes a schedule makespan sum for each mapping. In the second phase, the FU $\Rightarrow$ PE mapping is fixed, and the DSE software generates all FU priority com-

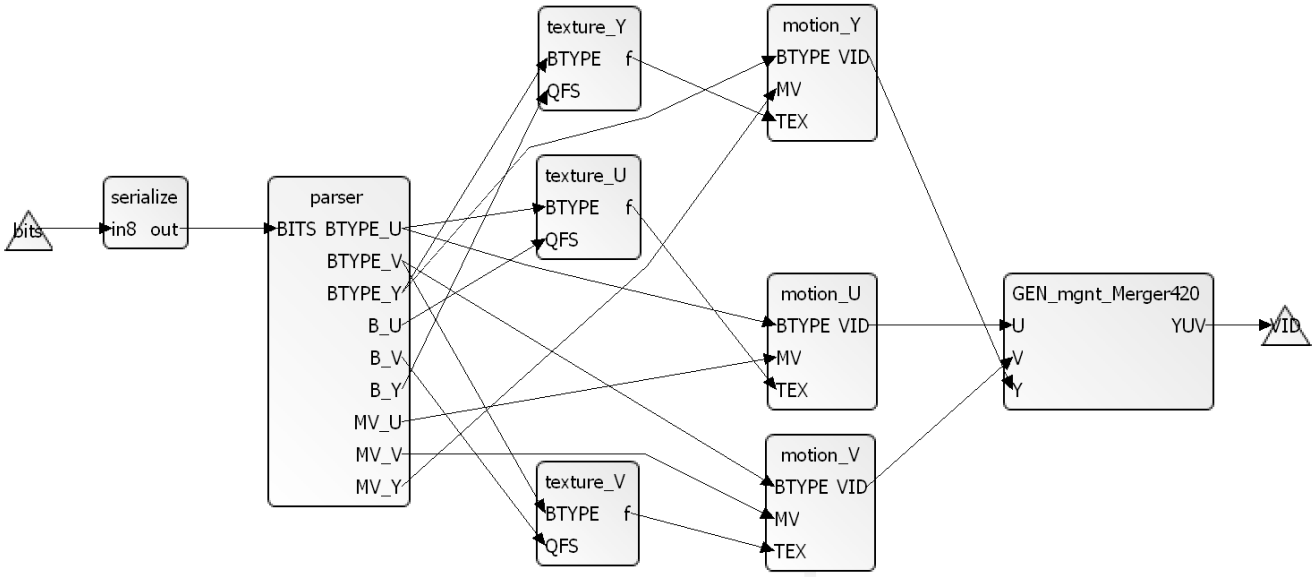

Figure 5. A high-level view of the RVC MPEG-4 SP decoder.

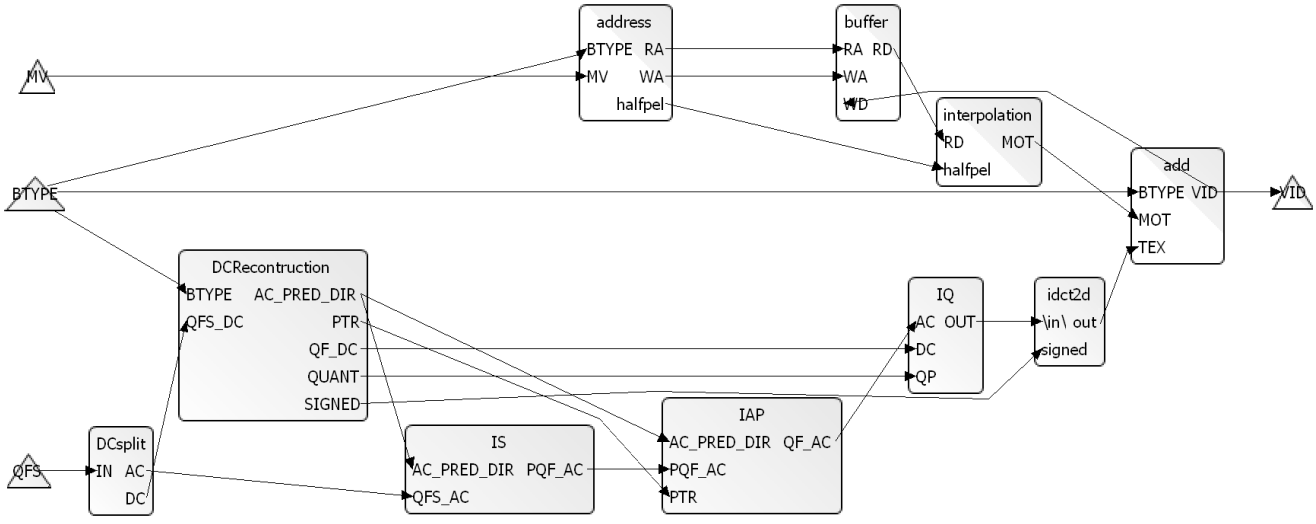

Figure 6. Motion compensation and texture decoding of RVC MPEG-4 SP.

Table 3. Latencies assigned to actions. Unit latency means that the action does not exist in the C code version of RVC MPEG-4 SP. Weight stands for the total computation time taken by the FU when executing a *hybrid block*.

Actor	Action	Latency	Weight
Address	done	1	48%
	init	2113	
	getmvx	192	
	getmvy	709	
	getw	353	
	geth	825	
	cmd.neither	680	
	cmd.noMotion	357	
	cmd.newVop	146	
	cmd.motion	174	
	read_addr	1084	
	write_addr	768	
Framebuf.	read	214	11%
	write	222	
Interpol.	start	1	5%
	row_col_0	162	
	other	162	
	done	1	
Add	cmd.newVop	342	4%
	cmd.textureOnly	174	
	cmd.motionOnly	174	
	cmd.other	174	
	combine	177	
	motion	76	
	texture	73	
	done	1	
IQ	get_qp	189	4%
	ac	189	
	done	1	
IDCT	transform (8x1)	3820	28%
Transpose	transpose (8x8)	6505	
Clip	limit	129	
	read_signed	156	
		avg=491	$\sum$=100%

binations, and computes the schedules makespan sums for each priority combination. In the third phase the FU $\Rightarrow$ PE mapping and FU priorities are both fixed and schedules are generated for visual inspection by Gantt-charts.

It is not guaranteed that this procedure produces absolutely minimal makespans, because only a fraction of the search space is explored. However, we have a reason to assume that the results are fairly close to optimal, because the first phase of the optimization produces considerably larger variations in makespan than the second phase. The makespan variations due to FU priorities are in the magnitude of +/- 1%.

The scheduling of the MPEG-4 SP HSDF graphs is performed by a basic scheduling algorithm that greedily fires HSDF actors as soon as they have enough input tokens to fire. However, the aforementioned FU priorities are observed, so that those actors that belong to a higher-priority FU, are fired before the actors that belong to lower-priority FUs.

The transformation tool that produces the input graphs for our DSE tool, currently generates HSDF graphs of all CAL actors in Figure 6, except IAP (Inverse AC prediction), IS (Inverse scan), DCReconstruction and DCSplit. There are different versions of the IAP actor for luma and chroma block decoding, which is currently not supported by our scheduling tool, although there is no such theoretical restriction. IS, DCReconstruction and DCSplit are not quasi-statically scheduled, because the IS actor executes different actions based on the values of tokens it acquires from the DCReconstruction actor. Thus, quasi-static scheduling of the IS actor (and its predecessors DCSplit and DCReconstruction) would require changing the schedule according to the token values arriving to IS. Whether this is feasible or not, is not sure. Finding this out is a clear direction for future work.

4.2. The results

A separate FU $\Rightarrow$ PE mapping was acquired for each IPC cost value that was imposed on the system. In the experiments we searched solutions for IPC costs between 100 and 800. To give a meaning to the IPC cost, it is worthwhile to mention that the average action latency was found out to be 491.

When the IPC cost was 100, the DSE algorithm mapped the tasks to four processors (A, B, C and D), such that Address, Framebuffer and IDCT2D had dedicated processors, and the rest of the actors were mapped to one processor. This can be seen in Figure 7 on column one.

When the IPC cost was increased to 200, the DSE software again found the best optimization result from a 4-PE solution. This time Address, Framebuffer and Interpolation had dedicated PEs, whereas IQ, IDCT and Add were

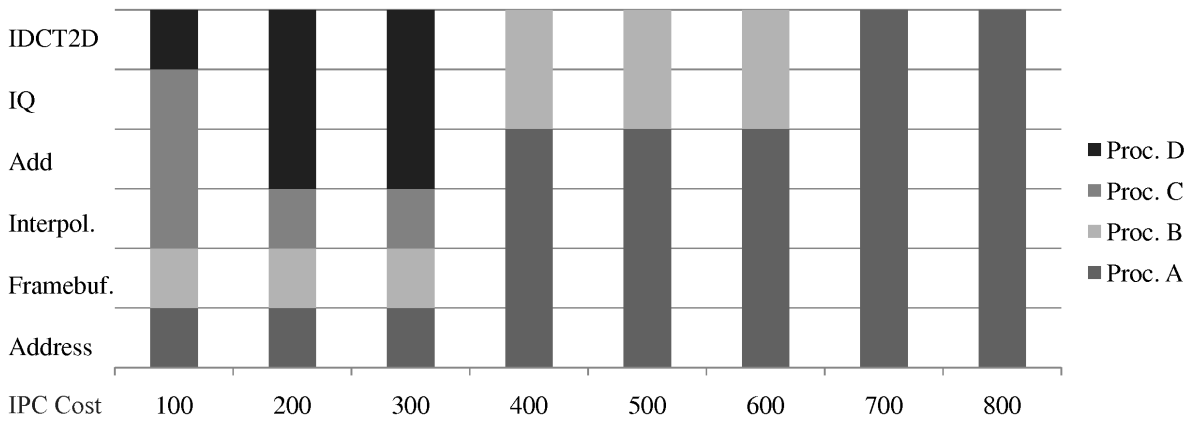

Figure 7. The FU ⇒ PE mapping as a function of IPC cost.

mapped to PE D. After increasing the IPC cost to 400, the computations were mapped on two PEs. Curiously, this mapping reflects the same motion compensation / texture decoding division that is also present in the original CAL models, but which has disappered from the HSDF graphs that are used as input to the DSE tool. Finally, when the IPC cost becomes higher than 700, a uniprocessor implementation provides the best throughput: everything is computed on PE A.

Figure 9 shows the schedule makespan of the mappings as a function of IPC cost. The two-processor solutions (IPC cost = [400, 700[) do not offer a considerable performance advantage over the one-processor solution, which can be explained by the utilization of the inverse quantization (IQ) and IDCT operations: only two operation modes out of five invoke actions of these FUs. Figure 8 shows the Gantt-chart of the 2-PE system (IPC cost = 400) for the *hybrid block* mode. PE B has a utilization of around 50% in this mode, which is not very good. Other mappings that would balance the 2 PEs better, are made unusable by the high IPC cost.

The 4-PE mappings, also visible in Figure 8, reduce the makespan nicely when compared to the 1-PE and 2-PE solutions. However, using this many PEs is only sensible when the IPC cost is low. The results point out that successful mapping of the RVC MPEG-4 SP decoder to a multiprocessing system requires low-latency communication between the processing elements in the system.

The topmost Gantt-chart in Figure 8 also shows that the Address actor is a bottleneck: the dedicated PE A is fully utilized and other PEs need to wait for PE A to provide data. Naturally, this can not be generalized: more efficient implementations of the Address actor and/or a processor other than NIOS II/s might change the situation. This can also be seen in the rightmost column of Table 3, where the total computation time of each FU is shown as a percentage value.

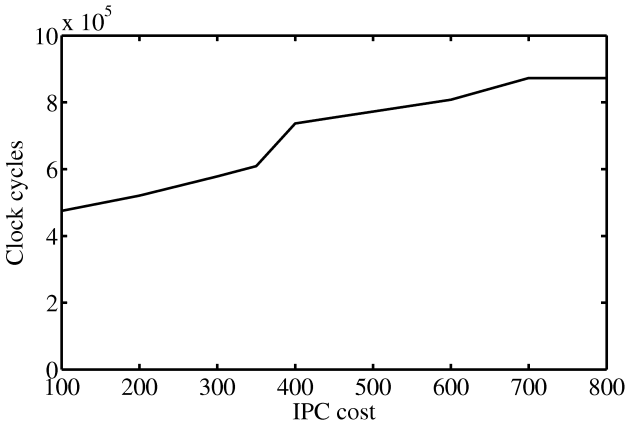

Figure 9. Schedule makespan for the *hybrid block* operation mode as a function of IPC cost.

5. Conclusion

In this paper we have presented the results of design space exploration for finding a good mapping of RVC MPEG-4 Simple Profile decoder functions to a multiprocessor system. Depending on the magnitude of the interprocessor communication cost, the design space exploration software maps the functions from one to four processors.

Prior to mapping the functions to processors, the CAL language specification of the RVC MPEG-4 decoder has been transformed to a set of quasi-statically schedulable HSDF graphs. The mapping of functions to processors has been done for these HSDF graphs.

A graphical user interface is currently being constructed for the DSE tool and it has been integrated to the transformation tool that produces HSDF graphs from CAL networks. As future work, the multiprocessor schedules and

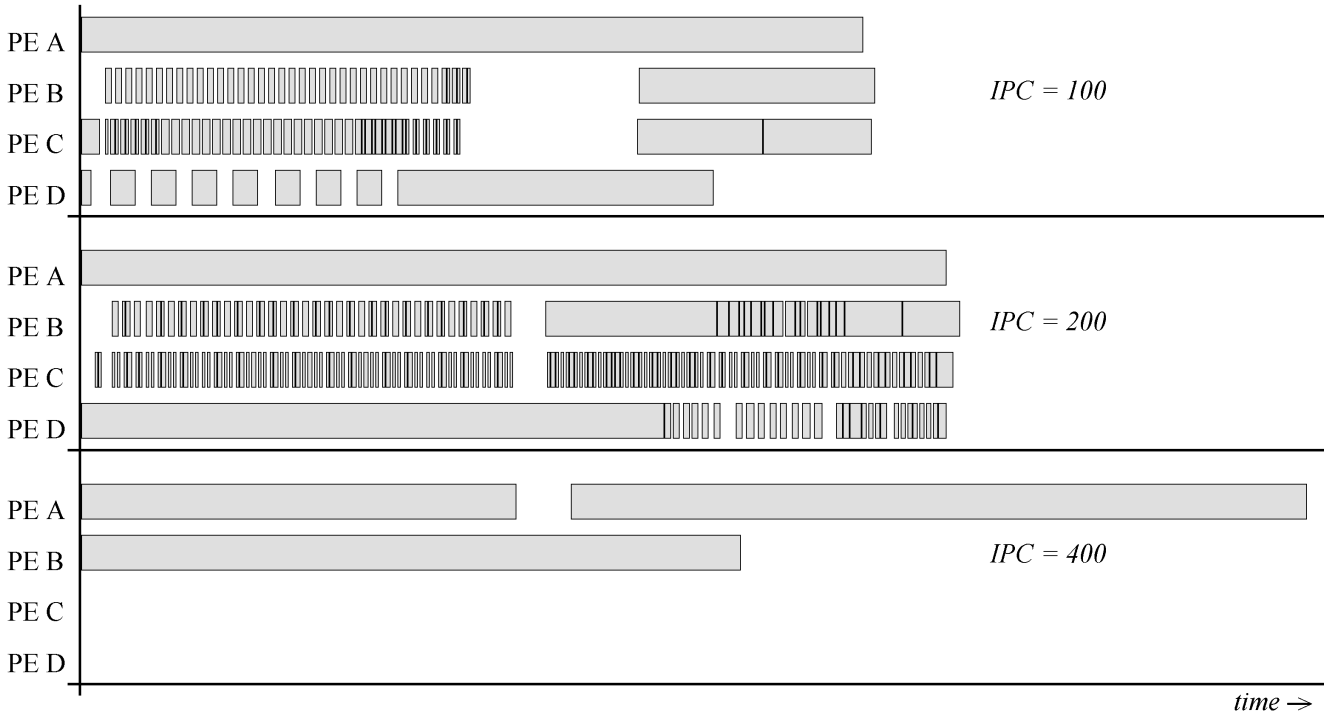

Figure 8. Schedules for the *hybrid block* operation mode. Generally, several actors execute on each PE in an interlaced order. To see which actors are mapped to each PE, see Figure 7.

mappings pointed out by the DSE tool are to be tested on a real multiprocessor system.

References

[1] Altera. Altera corp. Nios II processor reference handbook: http://www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf, 2009.

[2] S. S. Bhattacharyya, G. Brebner, J. W. Janneck, J. Eker, C. von Platen, M. Mattavelli, and M. Raulet. OpenDF - a dataflow toolset for reconfigurable hardware and multicore systems. Technical Report BTH-00422, Dept. of Systems and Software Engineering, Blekinge Institute of Technology, 2008.

[3] J. Boutellier, C. Lucarz, S. Lafond, V. Martin Gomez, and M. Mattavelli. Quasi-static scheduling of CAL actor networks for Reconfigurable Video Coding. *Journal of Signal Processing Systems*, page (to appear), 2009.

[4] J. Eker and J. W. Janneck. CAL language report. Technical Report UCB/ERL M03/48, UC Berkeley, 2003.

[5] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity - the Ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, January 2003.

[6] J. Gorin, J.-F. Nezan, M. Raulet, and M. Wipliez. Open RVC-CAL Compiler, http://sourceforge.net/projects/orcc, 2009.

[7] E. A. Lee. *VLSI Signal Processing III: Recurrences, Iteration, and Conditionals in Statically Scheduled Block Diagram Languages*, pages 330–340. IEEE Press, 1988.

[8] E. A. Lee and D. G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, September 1987.

[9] C. Lucarz, M. Mattavelli, M. Wipliez, G. Roquier, M. Raulet, J. W. Janneck, I. D. Miller, and D. B. Parlour. Dataflow/actor-oriented language for the design of complex signal processing systems. In *Conference on Design and Architectures for Signal and Image Processing*, pages 168–175, Bruxelles, Belgium, November 2008.

[10] MPEG. ISO/IEC FCD 23002-4 Information technology – MPEG video technologies – Part 4: Video tool library, 2008.

[11] H. Oh and S. Ha. Hardware-software cosynthesis of multi-mode multi-task embedded systems with real-time constraints. In *CODES '02: Proceedings of the tenth international symposium on Hardware/software codesign*, pages 133–138, Estes Park, CO, 2002.

[12] J. L. Pino, S. S. Bhattacharyya, and E. A. Lee. A hierarchical multiprocessor scheduling system for DSP applications. In *Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 122–126, Pacific Grove, CA, 1995.