Development and optimization of high level dataflow programs: the HEVC decoder design case

Khaled Jerbi, Daniele Renzi, Damien de Saint-Jorre, Hervé Yviquel, Mickaël Raulet, Claudio Alberti, Marco Mattavelli

Development and optimization of high level dataflow programs: the HEVC decoder design case

Khaled Jerbi*, Daniele Renzi†, Damien De Saint Jorre†, Hervé Yviquel*,
Mickaël Raulet*, Claudio Alberti†, Marco Mattavelli†
*IETR, INSA Rennes, CNRS UMR 6164, UEB
20, Av. des Buttes de Coësmes, 35708 Rennes
Email: firstName.name.@insa-rennes.fr
†Microelectronic Systems Lab, EPFL CH-1015
Lausanne, Switzerland
Email: firstName.name@epfl.ch

***Abstract*—With the standardization of the new High Efficiency Video Coding (HEVC) compression algorithm, a dataflow specification of the HEVC decoding process is also available as part of the standard. This paper presents methodologies to improve and optimize the performance of implementations derived by the dataflow specification. Regarding the architectural aspect of dataflow network, the throughput has been increased by developing more potential parallelism. For the platform aspect, critical processes have been optimized by applying SIMD functions and communications have been improved by cache efficient FIFO implementation.**

Results revealed an average acceleration factor of 7 in the decoding framerate over the reference dataflow implementation.

I. Introduction

The availability of high resolution screens supporting 4K and 8K Ultra High Definition TV formats, has raised the requirements for better performing video compression algorithms. With this objective MPEG and ITU have recently finalized the development of the new High Efficiency Video Coding (HEVC) video compression standard [1] successfully addressing these demands in terms of higher compression and increased potential parallelism when compared to previous standards. So as to guarantee real-time processing for such extremely high data rates, exploiting the parallel capabilities of recent many/multi-core processing platforms is in most of the cases an obliged implementation option for both encoders and decoders. In this context, dataflow programming is a particularly attractive approach because its intrinsic properties provide the portability of the potential parallelism on different processing platform.

The MPEG-RVC framework [2] is an ISO/IEC standard conceived to address these needs. It is essentially constituted by the RVC-CAL actor dataflow language [3] and a network language, and aims at replacing the traditional monolithic standard specification of video codecs with a dataflow specification that better satisfies the implementation challenges. The library of actors is written in RVC-CAL and provides the components that are configured using the network language to build a dataflow program implementing an MPEG decoder.

The main contributions of this work are: a) the development of an RVC-based dataflow program implementing the HEVC decoder; b) the optimization of the dataflow by exposing an higher level of potential parallelism; c) the optimization of the program for the mapping on x86 architectures using SIMD functions and efficient FIFO cache implementations. The paper is organized as follows: in Section II, an overview on the RVC framework and the dataflow HEVC decoder developed according to the RVC formalism is presented. Section III, details the methodologies used to profile and to improve the performance of the decoder. Finally, Section IV shows the implementation results on multi-core software platform.

This work is done as part of 4EVER, a French national project with support from Europe (FEDER), French Ministry of Industry, from French Regions of Brittany, Ile-de-France and Provence-Alpes-Côte-d'Azur, from Competitivity clusters Images & Reseaux (Brittany), from Cap Digital (Ile-de-France), and from Solutions Communicantes Securisées (Provence-Alpes-Côte-d'Azur).

II. Background

The emergence of massively parallel architectures, along with the need for modularity in software design, has revived the interest in dataflow programming. Indeed, designing processing systems using a dataflow approach presents several advantages when dealing with complex algorithms and targeting parallel and possibly heterogeneous platforms.

A. Reconfigurable Video Coding

The MPEG-RVC framework is an ISO/IEC standard aiming at replacing the monolithic representations of video codecs by a library of components. The framework allows the development of video coding tools, among other applications, in a modular and reusable fashion by using a dataflow programming approach. RVC presents a modular library of elementary components $(actors)$. An RVC-based design is a dataflow directed graph with actors as vertices and unidirectional FIFO channels as edges. An example of a graph is shown in Figure 1. Every directed graph executes an algorithm on sequences of tokens read from the input ports and produces sequences of tokens in the output ports.

Actually, defining several implementations of video processing algorithms using elementary components is very easy and fast with RVC since every actor is completely independent

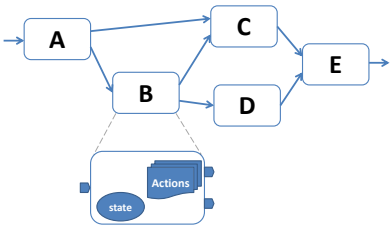


Fig. 1: A dataflow network of five processes, the vertices named from A to E, that communicate through a set of communication channels, represented by the directed edges.

from the rest of the other actors of the network. Every actor has its own scheduler, variables and behavior. The only way of communication of an actor with the rest of the network are its input ports connected to the FIFO channels to check the presence of tokens. Then, an internal scheduler enables or not the execution of elementary functions called actions depending on their corresponding firing rules. Thus, RVC ensures concurrency, modularity, reuse, scalable parallelism and encapsulation. To manage all the presented concepts of the standard, RVC presents a framework based on the use of a subset of the CAL actor language called RVC-CAL that describes the behavior of the actors.

The RVC framework is supported by a set of tools such as the Open RVC-CAL Compiler (Orcc). Orcc[1] [4] is an open-source toolkit dedicated to the development of RVC applications. Orcc is a complete Eclipse-based IDE that embeds two editors for both actor and network programming, a functional simulator and a dedicated multi-target compiler. The compiler is able to translate the RVC-based description of an application into an equivalent description in both hardware [5], [6] and software languages [7] for various platforms (FPGA, GPP, DSP, etc). A specific compiler back-end has been written to tackle each configuration case such as presented in Figure 2.

B. Dataflow-based HEVC decoder

HEVC is the last born video coding standard, developed conjointly by ISO and ITU, as a successor to AVC / H.264. HEVC is improving the data compression rate, as well as the image quality, in order to handle modern video constraints such as the high image resolutions 4K and 8K [1]. Another key feature of this new video coding standard is its capability for parallel processing that offers scalable performance on the trendy parallel architectures.

Such parallel capabilities offer a great opportunity to show the merits of the RVC approach. Consequently, the RVC working group has developed, in parallel with the standardization process, an implementation of the HEVC decoder using the RVC framework, which is presented in Figure 3. The description is decomposed in 4 main parts:

1) the parser: it extracts values needed by the next processing from the compressed data stream so called *bitstream*. The stream is decompressed with entropy decoding techniques, then the syntax elements composing the stream are extracted in order to be transmitted to the actors that they may concern. The parser applies a Context-adaptive binary arithmetic coding $(CABAC)$ to extract the syntax element of the bitstream.
2) the residual: it decodes the error resulting of the image prediction using Inverse integer Transform (IT), which is no other than an integer implementation of the well-known IDCT [8]. The transform allows spatial redundancy reduction within the encoded residual image. As presented in figure 3, the IT can be applied on different blocks sizes (4x4 .. 32x32) and the dataflow description allows parallelizing the processes.
3) the prediction part: it performs the intra and inter prediction. Intra prediction is done with neighbouring blocks in the same picture (spatial prediction) whereas inter prediction is performed as a motion compensation with other pictures (temporal prediction). The inter predication also implies the use of a buffer, known as Decoding Picture Buffer (DPB), containing decoding pictures, needed to perform the temporal prediction.
4) the filter: it is used to reduce the impact of the prediction

[1]Orcc is available at http://orcc.sf.net

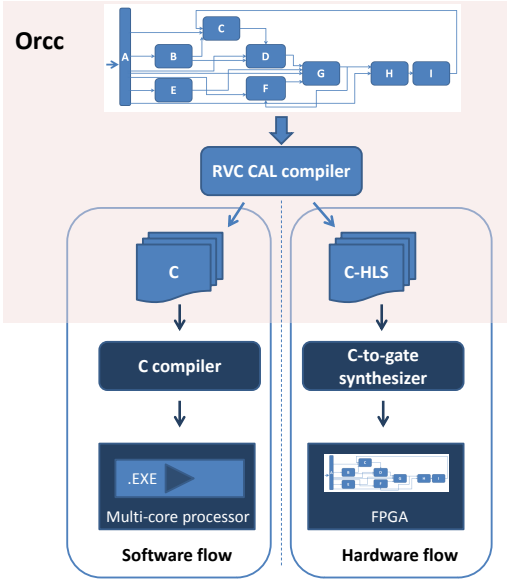


Fig. 2: Multi-target compilation infrastructure

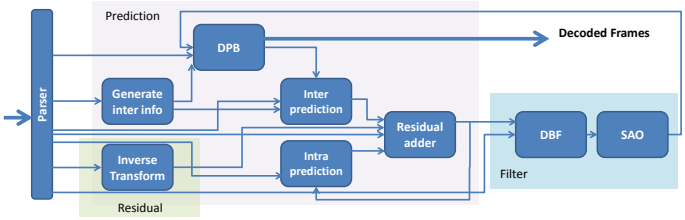


Fig. 3: Standard RVC specification of the MPEG HEVC decoder

on the image rendering. This part contains two different filters. On the one hand, the DeBlocking Filter (DBF) [9] is used to smooth the sharp edges between the macro-blocks. On the other hand, the Sample Adaptive Offset filter (SAO) [10] is used to better restore the original signal using an offset look up table.

III. Optimization of the RVC-based HEVC decoder

In order to assess the performance of the dataflow HEVC decoder presented above, Orcc has been used to generate a C implementation. The generated project is compiled with GCC and executed on a Xeon CPU at 3,2 Ghz. The preliminary results on HD streams showed a low throughput of 6.1 Frames/second. The mapping of the actors on multi-core for parallel execution did not bring scalable results.

A. Profiling

In order to better understand the bottlenecks of the design, profiling tools have been used to evaluate the workload of each actor and the obtained results have been reported in Figure 4. Results show that only 3 actors (Inter Prediction, DPB and SAO filter) consume 60% of the whole workload which means that these actors require an optimization stage and a refactoring to expose an higher potential of parallelism. In the following, an architecture optimization based on the split of the decoding process into luminance component (Y) and chrominance components (U and V) separately is presented. Then, optimization methodologies dedicated to x86 platforms are introduced.

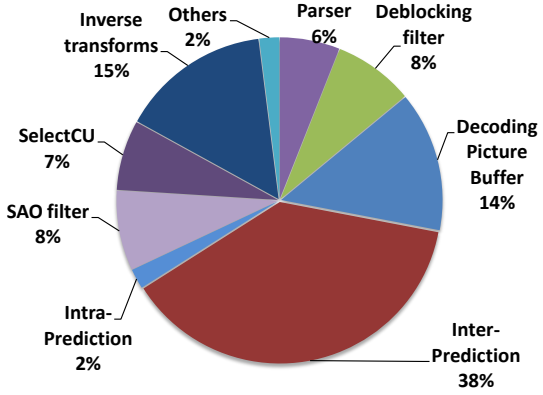


Fig. 4: Actors workload of the HEVC decoder

B. Architecture optimizations

In the first version of the decoder, sequential decoding of the image luminance and chrominance components was applied. This description is changed to split the processes into independent actors for each image component as illustrated in Figure 5. The application of this transformation had a direct impact on the workload as shown in Figure 6 where most of the critical actors workloads became close to the rest of the design, such as 6% for the DPB-Y and 11% for SAO-Y. Concerning the Luminance component of the Inter Prediction, a 23% is still considered to be a major bottleneck. In the following,a local optimization of this actor by linking with optimized functions from MPEG libraries has been applied.

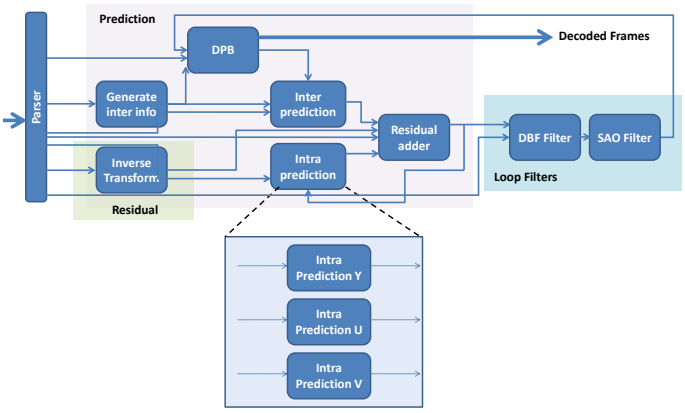


Fig. 5: YUV split of the HEVC decoder: example of split of the intra decoder at finer granularity; the same split is applied on most actors.

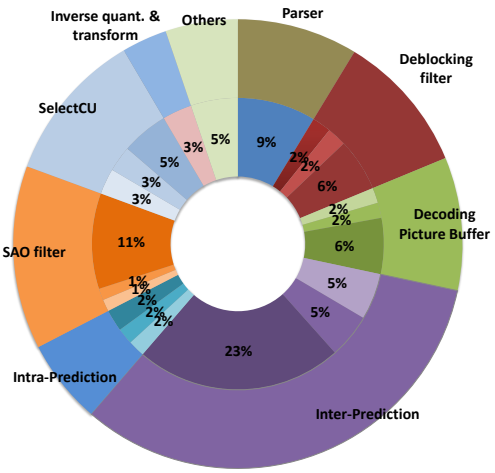


Fig. 6: Actors workload of Post YUV split

C. Platform-specific optimizations

Beside the possibility of using different dataflow network structures, the standard RVC dataflow program can also be implemented with platform-specific optimizations. In particular, a new methodology to use platform-specific optimized kernels to accelerate the internal processing of actors (i.e. actions) has been introduced, and an optimized FIFO channels implementation to speed-up the communication between processor cores that share a common cache memory has been developed.

1) Optimized kernels: Considering the compiler limitations to perform low-level optimization on high-level code, we propose a new technique to insert optimized architecture-specific kernel code within high-level descriptions of dataflow

application. In this work, we used Intel SIMD instructions to target x86 architectures. Our techniques relies on an annotation mechanism in order to keep the portability of the high-level description over multiple platforms:

1) First, the developer identifies the code to optimize and move it in its own procedure, knowing that the optimized kernels have to use the same parameters than their equivalents in CAL. The optimized version should be available into an external library (such as FFMPEG).
2) Then, the developer adds the directive `@optimize` on top of the CAL procedure to identify the optimized version of the procedure (see Listing 1). The directive is based on the following syntax `@optimize(condition="CONDITION", name="NAME")` where NAME is the name of the optimized kernel and CONDITION is a predefined condition that enable the execution of the kernel.
3) Finally, the generated code can use the optimized kernels when they are available (see Listing 2).

```
@optimize(condition="defined(OPENHEVC_ENABLE)", name="
    put_hevc_qpel_h")
procedure put_hevc_qpel_h_cal(int(size=16) arg1[64*64],
    int arg2)
begin
  // Kernel body in CAL
  (...)
end
```

Listing 1: CAL code

```
void put_hevc_qpel_h_cal(i16 arg1[4096], i32 arg2) {
  #if defined(OPENHEVC_ENABLE)
  // Optimized kernel
  put_hevc_qpel_h(arg1, arg2);
  #else
  // Standard kernel
  (...)
  #endif
}
```

Listing 2: Generated code

As a result, optimized applications easily stay compatible with all backends and platforms.

To link with SIMD functions, the CAL code undergoes small modifications by adding annotations and by corresponding functions they become identical to SIMD ones (arguments number and types). As explained in Figure 7, the FFMPEG SIMD functions are compiled to obtain a dynamic library. Then using pretty printing, a correct link of the C project with the dynamic library is guaranteed.

2) Cache-efficient FIFO channels: In software, FIFO channels are traditionally implemented by a circular buffer allocated in shared memory. *Read* and *write* are then achieved by accessing the buffer according to read and write indexes that are updated afterwards. The state of FIFO channels is known by comparison of their indexes. Using circular buffer to implement FIFO channels avoids side shuffles of data after each reading, but implies an advanced management of memory indexes that may ultimately lead to poor performance. In modern general-purpose processors, the processor cores usually communicates through common shared memory accessed using cache mechanism. Naive implementation of FIFO channels (see Figure 8-a) results in cache inefficiency because of false sharing. As a result, a memory padding is added on each FIFO index [11] in order not to share the same cache line as explained in Figure 8-b.

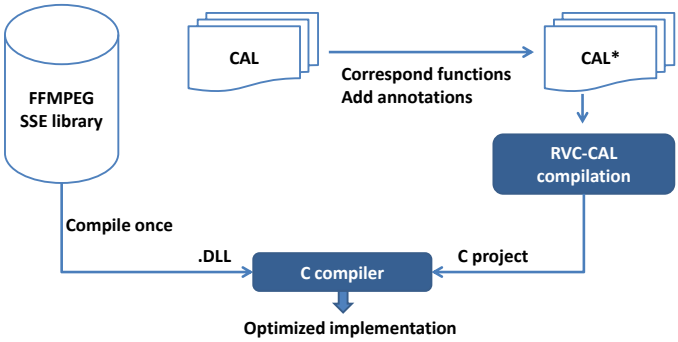


Fig. 7: Conception flow of the SIMD linked implementation shows a compilation of a RVC-CAL code with annotations linked with a DLL library generated from the compilation of FFMPEG.

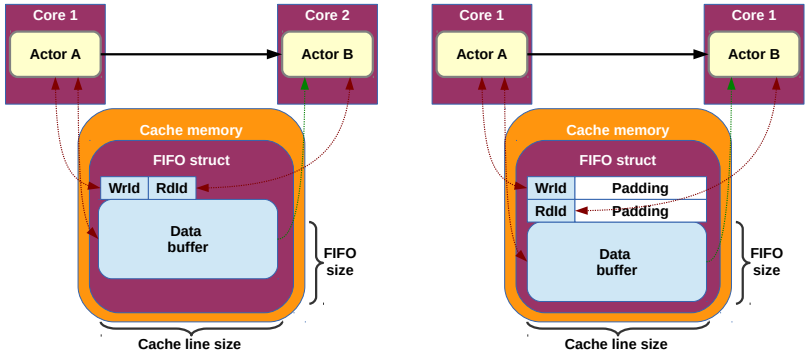
IV. Results

To apply the optimization methodologies evoked above, an Intel Xeon CPU at 3.2 GHz with 6 cores has been used. We applied the HD stream "Kimono" with Low Delay encode and QP=27. The HD stream "Kimono" with Low Delay encode and QP=27 were the test sequence selected for the experiments. In Table I, all the results of the reference design and the YUV design with and without x86 optimizations are presented.

TABLE I: Decoding framerates of the reference and the YUV designs on multi-core processor (in FPS) using all combinations of enabling and disabling SIMD functions and cache padding.

			Number of processor cores					
Description	Padding	SIMD	1	2	3	4	5	6
Reference	OFF	OFF	6,1	8,4	8,5	8,2	9,4	9,5
	ON	OFF	6,1	9	9	8,8	10,6	10,6
	OFF	ON	8,3	11,9	9,9	8,1	7,9	8,5
	ON	ON	8,3	13,1	14,8	15,6	15,3	15,1
Y U V	OFF	OFF	9,5	16,7	19	22,6	22,7	19
	ON	OFF	9,5	17	23,6	23,2	23,4	24,1
	OFF	ON	15,8	27,7	37,4	30,4	35,9	35,5
	ON	ON	15,8	28,8	40,5	43,2	46,6	39,1

To focus on the evolution of the framerate, the curves of Figure 9 are considered. The impact of the YUV design is the scalability with the number of cores. The framerate reaches a maximum of 46.6 FPS with 5 cores then starts decreasing and this is due to the increasing communication cost between processing cores. To compare the obtained result with a reference from the literature, the same stream on the OpenHEVC decoder [12] has been applied. Using one thread, OpenHEVC decodes at 59 FPS in mono-thread which is close



(a) Classic cache implementation of FIFO channels (b) Cache-efficient implementation of FIFO channels

Fig. 8: A smart FIFO implementation in the cache of the processor. In (a), the classic definition of the indexes results in setting indexes in the same line of cache which requires useless waits for refresh. In (b), by adding the paddings, the indexes are set into distinct lines of cache.

TABLE II: Decoding acceleration of the reference and the YUV designs on multi-core processor (in FPS) using all combinations of enabling and disabling SIMD functions and cache padding

			Number of processor cores					
Description	Padding	SIMD	1	2	3	4	5	6
Reference	ON	OFF	1	1.47	1.47	1.44	1,73	1,73
	OFF	ON	1.36	1.95	1.62	1.32	1.29	1.39
	ON	ON	1.36	2.14	2.42	2.55	2.50	2.47
Y U V	OFF	OFF	1.55	2.73	3.11	3.70	3.72	3.11
	ON	OFF	1.55	2.78	3.86	3.80	3.83	3.95
	OFF	ON	2.59	4.54	6.13	4.98	5.88	5.81
	ON	ON	2.59	4.72	6.63	7.08	7.63	6.40

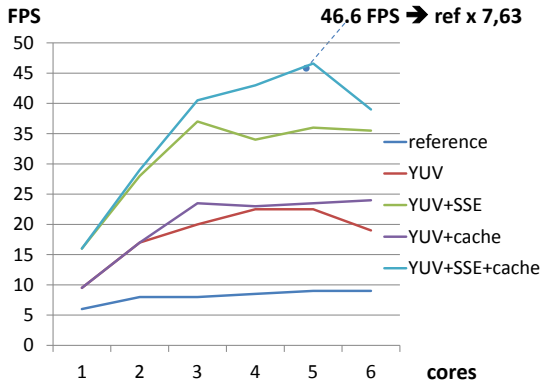


Fig. 9: Decoding framerate using the different optimization techniques compared with the reference description

to the performance of the automatically generated code from high-level description in dataflow.

V. Conclusion

In this paper, a dataflow description of the HEVC decoder based on the RVC framework has been presented. To improve the performance of the decoder, more parallelism in the architecture has been achieved by splitting most of the decoding processes into independent Y, U and V components separately. Platform specific x86 optimizations have been developed and they consist of using a smart FIFO cache implementation and substituting some critical functions with SIMD ones. Results show an acceleration that exceeds a factor of 7.6 which allows real time decoding for HD streams.

In the ongoing works, the achievement of an higher level of parallelism by applying frame-based decoding is under study. Some critical actors, considered as bottlenecks, are also under consideration for improving their processing efficiency.

References

[1] Gary J. Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1649–1668, December 2012.

[2] Marco Mattavelli, Mickaël Raulet, and Jörn W. Janneck. MPEG reconfigurable video coding. In Shuvra S. Bhattacharyya, Ed F. Deprettere, Rainer Leupers, and Jarmo Takala, editors, *Handbook of Signal Processing Systems*, pages 281–314. Springer, New York, NY, USA, 2013.

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

[4] Hervé Yviquel, Antoine Lorence, Khaled Jerbi, Alexandre Sanchez, Gildas Cocherel, and Mickaël Raulet. Orcc: Multimedia development made easy. In *Proceedings of the 21st ACM international conference on Multimedia*, 2013.

[5] Khaled Jerbi, Mohamed Abid, Mickaël Raulet, and Olivier Déforges. Automatic Generation of Synthesizable Hardware Implementation from High Level RVC-CAL Description. In *International Conference on Acoustics, Speech, and Signal Processing*, volume 2012, pages 1–20, 2012.

[6] Mariem Abid, Khaled Jerbi, Mickaël Raulet, Olivier Déforges, and Mohamed Abid. System Level Synthesis Of Dataflow Programs: HEVC Decoder Case Study. In *Electronic System Level Synthesis Conference (ESLsyn), 2013*, 2013.

[7] Matthieu Wipliez, Ghislain Roquier, and Jean-François Nezan. Software Code Generation for the RVC-CAL Language. *Journal of Signal Processing Systems*, 63(2):203–213, June 2009.

[8] N. Ahmed, T. Natarajan, and K.R. Rao. Discrete cosine transfom. *IEEE Transactions on Computers*, 23:90–93, 1974.

[9] A. Norkin, G. Bjontegaard, A. Fuldseth, M. Narroschke, M. Ikeda, K. Andersson, Minhua Zhou, and G. Van der Auwera. Hevc deblocking filter. *Circuits and Systems for Video Technology, IEEE Transactions on*, 22(12):1746–1754, 2012.

[10] Chih-Ming Fu, Ching-Yeh Chen, Yu-Wen Huang, and Shawmin Lei. Sample adaptive offset for hevc. In *Multimedia Signal Processing (MMSP), 2011 IEEE 13th International Workshop on*, pages 1–5, 2011.

[11] Patrick PC Lee, Tian Bu, and Girish Chandranmenon. A Lock-Free , Cache-Efficient Shared Ring Buffer for Multi-Core Architectures. In *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 2–3, 2009.

[12] Wassim Hamidouche, Mickaël Raulet, and Olivier Déforges. Parallel SHVC decoder: implementation and analysis. In *Multimedia and Expo (ICME), 2014 IEEE International Conference on*, 2014.