On the Development and Optimization of HEVC Video Decoders Using High-Level Dataflow Modeling

Khaled Jerbi, Hervé Yviquel, Alexandre Sanchez, Daniele Renzi, Damien de Saint Jorre, Claudio Alberti, Marco Mattavelli, Mickael Raulet

Noname manuscript No.
(will be inserted by the editor)

On the development and optimization of HEVC video decoders using high-level dataflow modeling

Khaled Jerbi · Hervé Yviquel · Alexandre Sanchez · Daniele Renzi · Damien De Saint Jorre · Claudio Alberti · Marco Mattavelli · Mickaël Raulet

Received: date / Accepted: date

Abstract With the emergence of the High Efficiency Video Coding (HEVC) standard, a dataflow description of the decoder part was developed as part of the MPEG-B standard. This dataflow description presented modest framerate results which led us to propose methodologies to improve the performance. In this paper, we introduce architectural improvements by exposing more parallelism using YUV and frame-based parallel decoding. We also present platform optimizations based on the use of SIMD functions and cache efficient FIFOs. Results show an average acceleration factor of 5.8 in the decoding framerate over the reference architecture.

1 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 becoming compulsary for implementing both encoders and decoders. In this context, dataflow programming is a particularly attractive approach because its intrinsic properties allow natural decomposition for parallel platforms.

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: 1) the development of an RVC-based dataflow program implementing the HEVC decoder; 2) the optimization of the dataflow architecture by exposing an higher level of potential parallelism; 3) the optimization of the program for the execution on x86 architectures using SIMD functions and efficient FIFO cache implementations. The paper is organized as follows: in Section 2, we present an overview on the RVC framework. Section 3 details the dataflow HEVC decoder developed according to the RVC formalism. Section 4, details the methodologies used to improve the performance of the decoder. Finally, Section 5 shows the implementation results on multi-core software platform.

Jerbi, Yviquel, Raulet, Sanchez
IETR/INSA Rennes, CNRS UMR 6164, UEB
20, Av. des Buttes de Coesmes, 35708 Rennes
E-mail: firstName.name@insa-rennes.fr

Renzi, De Saint Jorre, Alberti, Mattavelli
Microelectronic Systems Lab, EPFL CH-1015
Lausanne, Switzerland
E-mail: firstName.name@epfl.ch

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).

2 Reconfigurable Video Coding

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.

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.

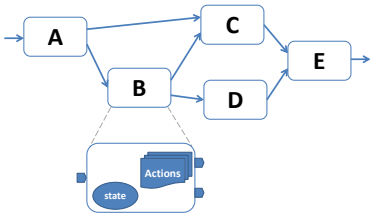


Fig. 1: A dataflow network of five processes, the vertices named from A to E, that communicate through a set of communication channels, represented by the directed edges.

Actually, defining several implementations of video processing algorithms using elementary components is very easy and fast with RVC since the internal state of every actor is completely independent from the rest of the 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 [7,8] 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.

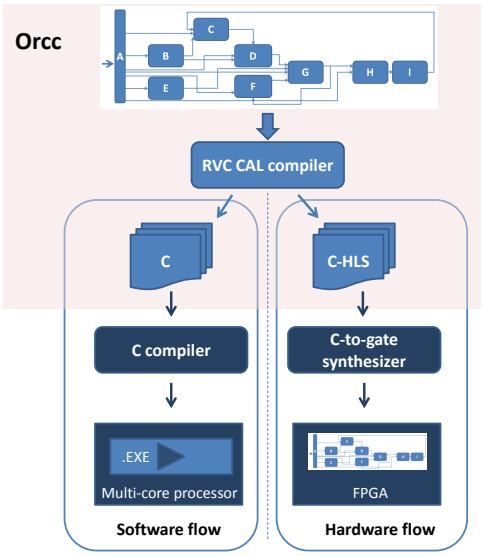


Fig. 2: Multi-target compilation infrastructure

3 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.

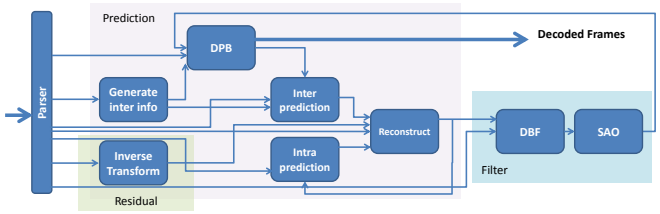


Fig. 3: Standard RVC specification of the MPEG HEVC decoder

3.1 Reference design

With the parallel capabilities, HEVC offers 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. 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 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.

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

3.2 Design profiling

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 1080x1920 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.

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.

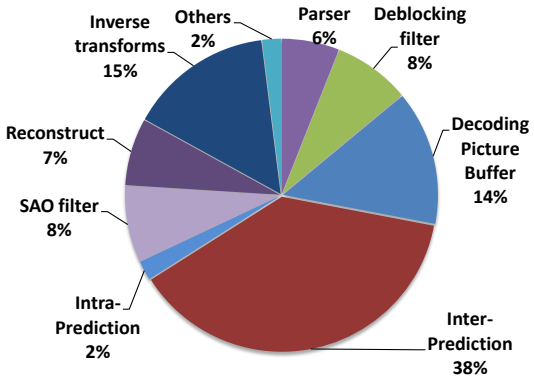


Fig. 4: Actors workload of the HEVC decoder

4 Methodologies for performance improvement

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. The same architecture is duplicated to enable a frame-based parallel decoding of the frames. Then, optimization methodologies dedicated to x86 platforms are introduced.

4.1 Architecture optimizations

4.1.1 YUV components parallel decoding

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 impact of such architecture is detailed by Weiwei et. al in [11]. Only the parser could not be split since the 3 components are put sequentially in the bitstream. Splitting the parser results in complex actors since the state of the CABAC has to be shared between those actors. The

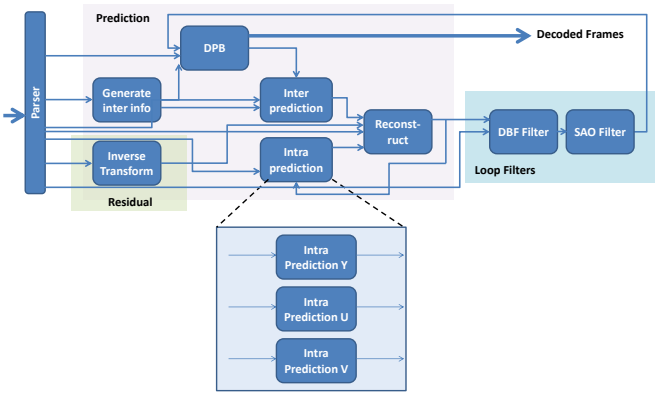


Fig. 5: YUV split of the HEVC decoder: example of split of the intra decoder at finer granularity; the same split is applied on most actors.

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.

4.1.2 Frame-based parallel decoding

Thanks to the modularity of dataflow modeling, the frame-based parallelization is theoretically, a duplication of the whole decoding actors. The principle is that the compressed stream is sent to n decoders and, following a selection strategy, a decoder is going to decode some frames and bypass others that are decoded by other decoders. Figure 7 shows the frame-based architecture where a set of decoders coexist. Some data

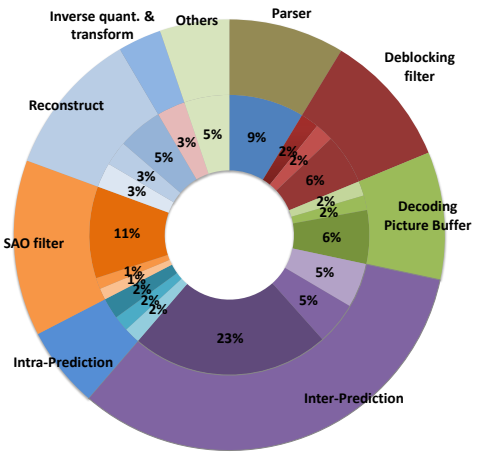


Fig. 6: Actors workload of Post YUV split

from a frame is needed to decode another frame such as motion vectors and reference lists. This problem was raised by Zhou et. al in [12] For that reason, a common parser and motion vector generator is added to communicate a correct stream of common data to the decoding picture buffer and also to the inter prediction actors of all decoders. The frame-based does not exculde the YUV split which means that it is possible to use N decoders all containing YUV components parallel decoding. In the following, we assume that FBn is a frame-based architecture with n decoders.

Figure 8 presents the distribution of the computational load within the software implementation of the frame-based description composed of 2 decoders (FB2).

The results show that the distribution is consistent among the tested implementations, as well as among the distribution observed in more traditional implementations [13],[14]. Motion compensation performed within the *Inter-predition* and the *Decoded Picture Buffer* is the most consuming part with about 50% of the computation load. Loops filters represent about 25% of the computation load. Entropy decoding and inverse quantization, perform both in the *Parser*, contributes to 10% of the global computation load on average. In FB2 design, this load increases to 17% since the parsing is duplicated in the *Main Parser* and each *Parser*. The *Inverse Transforms* represent between 3% to 12% of the computation load depending on code optimization of the designs. The last significant part, about 10% of the computation load, is specific to dataflow description: Known as *Reconstruction*, it adds the residual to the predicted blocks.

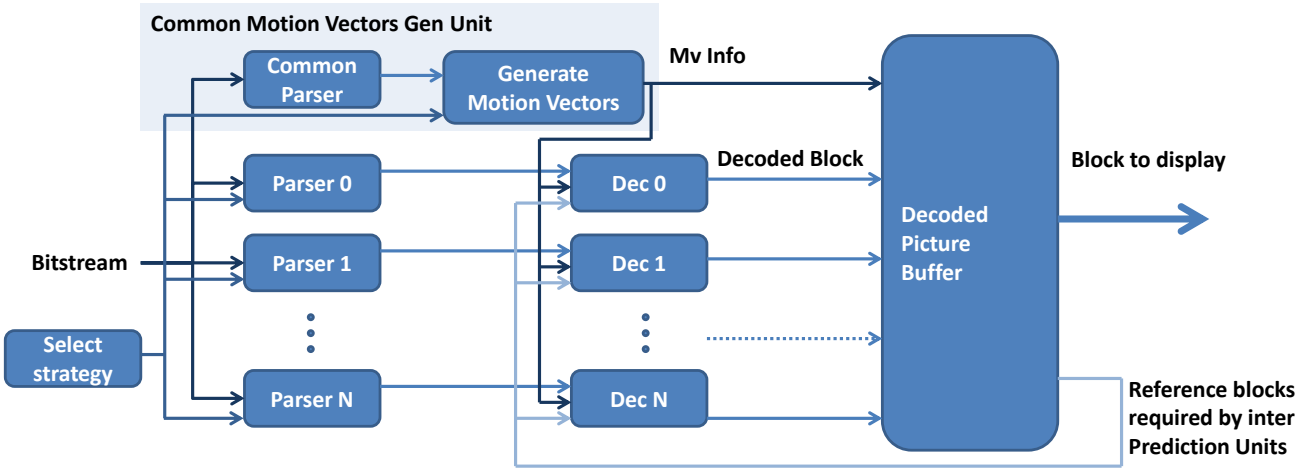


Fig. 7: Data flow frame-based design of HEVC decoder

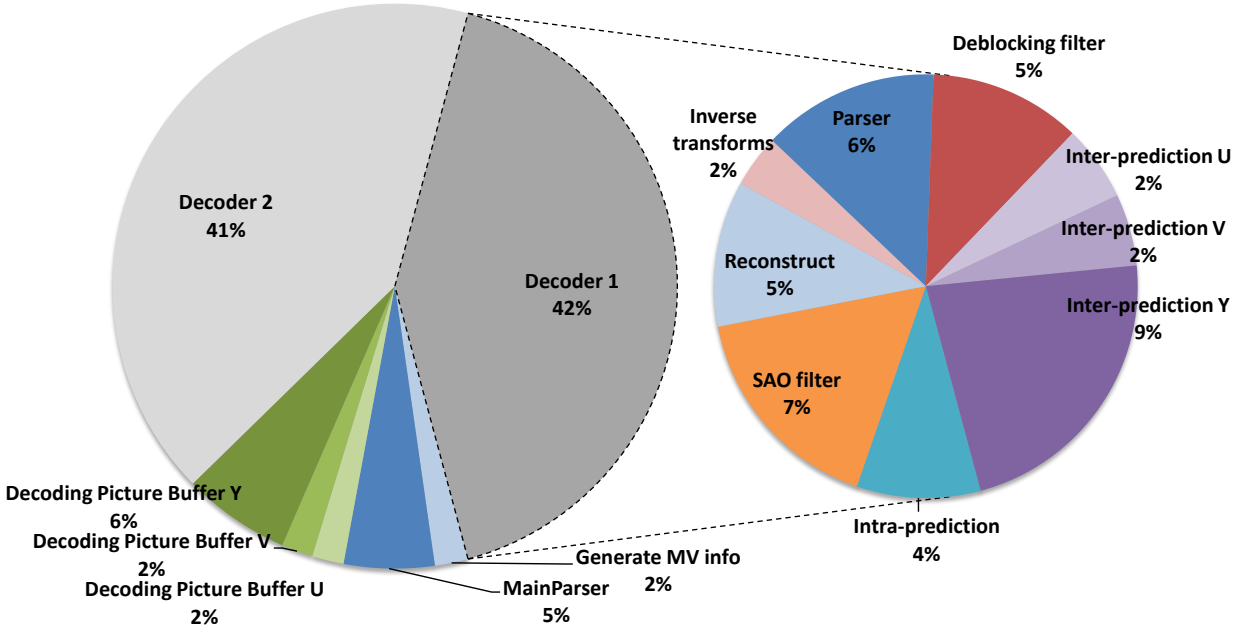


Fig. 8: Average load distribution in 3 dataflow HEVC implementations: *Kimono*, Low Delay, all QP values.

4.2 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.

4.2.1 Optimized SIMD 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 9, the FFMPEG library is compiled to allow the external use of SSE functions. Then, a correct link of the C project with the dynamic library is guaranteed by our build system.

4.2.2 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. Indeed, the functions that read data from a predecessor actor and the functions that write data to a successor actor are completely independent which means that, following the model of computation, it is impossible to read and write data in the same FIFO at the same time. In modern general-purpose processors, the processor cores usually communicates through common shared memory accessed using cache mechanism. When accessing a line of cache a processor has to check if there are fields that have been changed. In that case, to be sure of using the last stored data, it makes a refresh of the line of cache. When the read and the write indexes are set into different lines of cache, the refresh of the line containg the read index is applied only when reading and the same for writing. Naive implementation of FIFO channels (see Figure 10-a) results in cache inefficiency because of false sharing. As a result, a memory padding is added on each FIFO index [15] in order not to share the same cache line as explained in Figure 10-b.

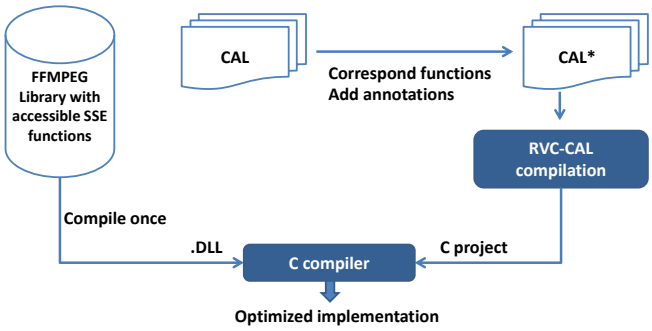


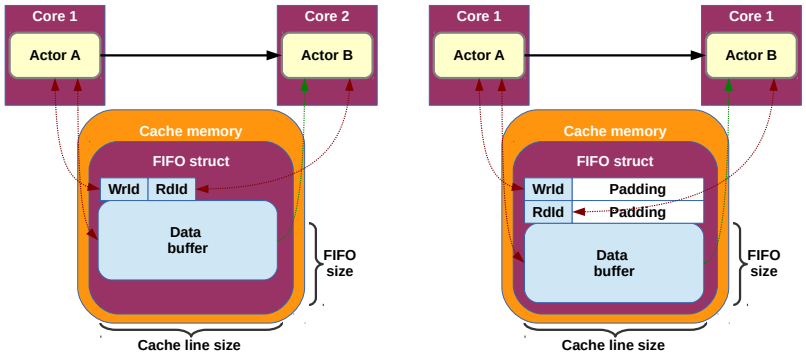
Fig. 9: Conception flow of the SIMD linked implementation shows a compilation of a RVC-CAL code with annotations linked with a DLL library generated from the compilation of FFMPEG.

5 Results

To apply the optimization methodologies evoked above, an Intel Xeon CPU at 3.2 GHz with 6 cores has been used. The experiments were applied on the streams of table 1:

Table 2 presents the decoding framerate of the YUV design on test streams using an increasing number of processor cores and 4 different encoding configurations (LD with QP=27 and 37, LP with QP=27 and 37).

To apply the frame-based design we reduced the configurations to QP=27 and LD encode since all 1080x1920 streams have almost equal results. This choice can be



(a) Classic cache implementation of FIFO channels

(b) Cache-efficient implementation of FIFO channels

Fig. 10: 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 1: Video sequences considered in the experiments

Sequences	Resolution	Framerate(FPS)
Kimono	1920x1080	24
BasketBallDrive		50
ParkScene		24
Cactus		50
BQTerrace		60
PeopleOnStreet	2560x1600	30
Traffic		30

Table 2: Decoding framerate of the YUV design on multi-core processor (in FPS)

			Number of processor cores					
Sequence info	Conf	QP	1	2	3	4	5	6
Kimono	LD	27	20	33	48	51	48	47
		37	27	46	54	65	85	83
	RA	27	22	34	50	53	54	52
		37	30	44	54	62	79	89
BasketBallDrive	LD	27	16	29	44	48	39	37
		37	25	37	51	58	85	82
	RA	27	17	27	38	49	44	39
		37	27	41	49	57	86	87
ParkScene	LD	27	17	26	35	37	43	43
		37	25	40	53	57	60	82
	RA	27	20	28	44	43	43	41
		37	28	46	55	61	67	81
Cactus	LD	27	19	30	37	47	46	48
		37	25	44	56	59	66	82
	RA	27	22	32	37	48	45	49
		37	27	45	59	60	69	89
BQTerrace	LD	27	19	27	38	42	40	39
		37	26	44	59	63	85	87
	RA	27	19	26	33	38	40	35
		37	29	47	62	67	86	90
PeopleOnStreet	LD	27	5	10	15	13	14	12
		37	9	16	20	18	25	23
	RA	27	7	13	18	18	16	16
		37	12	18	26	24	25	30
Traffic	LD	27	11	17	20	24	25	26
		37	15	24	32	34	38	37
	RA	27	13	21	24	25	28	26
		37	17	28	37	40	39	38

explained by the fact that a QP=27 is an average between QP=22 and QP=37 with a really good image quality and realistic bitrate for industry usage. We considered three designs: FB2, FB3, and FB4. All the framerate (FPS) results are summarized in Table 3. For all streams, the framerate decreases in the mono-core design. This is due to the increasing number of actors since we duplicate almost the whole decoder. Consequently, the global actors scheduler is slowing down the application when actor number grows.

For a better study of dataflow implementation of HEVC decoder, we analyzed the multi-threaded implementation of the OpenHEVC decoder. We could not use the HM reference because it does not support multi-threading or multi-core mapping. In the OpenHEVC implementation, the thread parallelism is mainly based on frames. With a predefined number of threads, the decoder assigns a frame for each thread. When there is a dependency from blocks that belong to another frame the threads are able to wait until the required block is decoded. Once a thread is free it starts a new frame. Table 4 presents the framerate obtained while decoding the same test streams with OpenHEVC on the same platform. Results showed that OpenHEVC is already much more efficient on single-core execution: This can be explained by the large use of assembly-level optimizations. It is also noticeable that the maximal performance is reached at 5 cores then remains stable. Since multi-threading enables the minimization of the communication cost on shared-memory architecture, the performance stays stable even if the maximal parallelism is already reached with fewer threads.

Table 3: Decoding framerate of the dataflow HEVC designs on multi-core processor (in FPS): Low Delay, and QP=27.

		Number of processor cores					
Sequence	Architecture	1	2	3	4	5	6
Kimono	Ref	11	16	20	23	22	19
	YUV	20	33	48	51	48	47
	FB2	18	29	47	54	64	53
	FB3	15	25	38	40	51	55
	FB4	14	20	35	42	44	59
BasketBallDrive	Ref	10	14	17	18	20	17
	YUV	16	29	44	48	39	37
	FB2	14	21	38	42	56	52
	FB3	11	16	21	25	32	35
	FB4	9	14	19	29	45	38
ParkScene	Ref	10	15	19	20	19	17
	YUV	17	26	35	37	43	43
	FB2	15	24	34	36	50	48
	FB3	12	19	28	34	35	42
	FB4	11	15	25	35	44	45
Cactus	Ref	12	16	18	21	22	20
	YUV	19	30	37	47	46	48
	FB2	18	24	34	42	60	50
	FB3	15	19	32	38	41	44
	FB4	13	18	27	35	55	52
BQTerrasse	Ref	10	15	19	20	22	21
	YUV	19	27	38	42	40	39
	FB2	17	23	35	36	32	49
	FB3	15	19	28	38	44	42
	FB4	11	16	23	37	51	58
PeopleOnStreet	Ref	4	6	9	11	12	10
	YUV	5	10	15	13	14	12
	FB2	4	8	11	12	21	14
	FB3	4	7	10	11	13	16
	FB4	3	5	8	12	14	12
Traffic	Ref	7	9	12	13	13	11
	YUV	11	17	20	24	25	26
	FB2	9	15	21	24	28	33
	FB3	7	12	18	23	22	19
	FB4	6	10	15	22	21	22

Table 4: Decoding framerate using OpenHEVC (in FPS): Low Delay, and QP=27.

	Number of threads					
Sequences	1	2	3	4	5	6
Kimono	59	96	136	145	144	144
BasketBallDrive	51	85	115	122	123	122
ParkScene	51	78	105	112	111	112
Cactus	63	102	142	152	150	151
BQTerrace	54	82	109	115	113	114
PeopleOnStreet	17	30	46	51	61	61
Traffic	33	51	73	77	91	92

For comparison, Figure 11 presents the framerate evolution with a various number of processor cores on the different designs. The FB2, FB3, and FB4 designs overstep the performance of the reference design respectively at 3, 4, and 5 cores. Which confirms that the more actors we have, the more cores we need to overcome the overhead of the scheduler. Due to the communications cost explained above, the FB2 design reaches a peak at 5 cores then it starts decreasing with more cores. FB3 and FB4 designs continue increasing with the 6 available cores. This framerate matches an acceleration peak of 5.81 as shown in Table 5. The rising curve of the framerate performance of FB3 and FB4 designs shows a potential continuous increase that can be established using more processing cores.

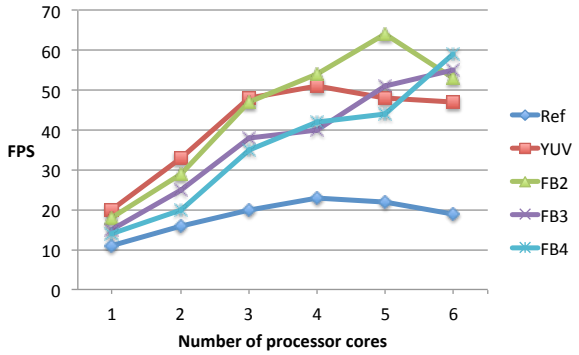


Fig. 11: Evolution of the decoding framerate with dataflow designs on multi-core platform: *Kimono*, Low Delay, and QP=27.

Table 5: Acceleration ratio: for the RVC designs between the multi-threaded design and the single-thread version of the ref design, and for OpenHEVC between its multi-threaded versus its single-threaded (Low Delay and QP=27)

		Number of threads				
Sequences	Design	2	3	4	5	6
Kimono	OpenHEVC	1.62	2.35	2.45	2.44	2.44
	YUV	1.81	3.00	4.36	4.63	4.36
	FB2	2.63	4.27	4.90	5.81	4.81
	FB3	2.27	3.45	3.63	4.63	5.00
	FB4	1.81	3.18	3.81	4.00	5.36
Traffic	OpenHEVC	1.54	2.21	2.33	2.75	2.78
	YUV	1.57	2.42	3.42	3.57	3.71
	FB2	2.13	2.98	3.42	3.99	4.71
	FB3	1.71	2.56	3.28	3.14	2.70
	FB4	1.41	2.13	3.14	2.98	3.14

6 Discussion

Considering the above results, the performance of the RVC decoder remains lower than the openHEVC which is expected since openHEVC is a native C application while the RVC decoder implementation is automatically generated. Basically, the purpose of this work is not to present a performance that oversteps a native application but to show that using a modular high level description it is possible to reach real time processing of high resolutions.

More generally, we underline that for high complexity applications RVC allows to go at least 3 times faster than C languages for example and even if the performance is lower than native codes, this framework is very attractive for developping complex applications that do not have hard real time constraints.

Moreover, the RVC-CAL code is target agnostic and current compilers allow the automatic generation of several languages such as C, C++, LLVM, C-HLS, Verilog etc, from a unique description. This feature is very important for hardware developpers since the validation of the application can be achieved in a software context and finally the hardware generation is performed so the prototyping goes faster.

7 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 and also by duplicating the decoding processes to enable a frame-based parallelism. 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 reaches a factor of 5.8 over the initial dataflow implementation standardized by MPEG, which allows real time decoding for 1080x1920 streams at 60 Hz. 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. H. Yviquel, A. Sanchez, P. Jaaskelainen, J. Takala, M. Raulet, and E. Casseau. Efficient software synthesis of dynamic dataflow programs. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 4988–4992, May 2014.
8. 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.
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. SHEN Weiwei, FAN Yibo, and ZENG Xiaoyang. A 64 cycles/mb, luma-chroma parallelized h. 264/avc deblocking filter for 4 k¡ cd0215f. gif¿ 2 k applications. *IEICE transactions on electronics*, 95(4):441–446, 2012.
12. J. Zhou, D. Zhou, J. Zhu, and S. Goto. A frame-parallel 2 gpixel/s video decoder chip for uhdtv and 3-dtv/ftv applications. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 23(12):2768–2781, Dec 2015.
13. Frank Bossen, Benjamin Bross, Karsten Sühring, and David Flynn. HEVC Complexity and Implementation Analysis. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1685–1696, 2013.
14. Marko Viitanen, Jarno Vanne, Timo D. Hämäläinen, Moncef Gabbouj, and Jani Lainema. Complexity Analysis of Next-Generation HEVC Decoder. In *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, pages 20–23, 2013.
15. 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.
16. Wassim Hamidouche, Mickaël Raulet, and Olivier Déforges. Parallel SHVC decoder: implementation and analysis. In *Multimedia and Expo (ICME), 2014 IEEE International Conference on*, 2014.