H.264/SVC decoder performance comparison for DSP-based consumer electronic applications

Fernando Pescador, Médéric Blestel, Eduardo Juarez, Mickaël Raulet, Matias Garrido

H.264/SVC DECODER PERFORMANCE COMPARISON FOR DSP-BASED CONSUMER ELECTRONIC APPLICATIONS

F. Pescador[1], M. Blestel[2], E. Juarez[1], M. Raulet[2] and M. Garrido[1]
[1]Universidad Politécnica de Madrid [2]Institut d'Electronique et de Télécommunications
[1]pescador, ejuarez, matias@sec.upm.es [2]mblestel, mraulet@insa-rennes.fr

ABSTRACT

In this paper, a performance comparison of two H.264/SVC-compliant video decoders, Joint Scalable Video Model and Open SVC, is presented. The performance, in terms of time spent to decode a stream, has been compared in PC and Digital Signal Processor (DSP) environments. The performance of the Open SVC decoder running on the PC is between three and eight times better than achieved with the JSVM decoder; while in the DSP environment, the improvement is between five and twelve times. These results show that the Open SVC decoder is more suitable as starting point for the implementation of embedded applications based on DSP[1].

1 INTRODUCTION

Nowadays, the terminals used in the multimedia broadcasting environments are heterogeneous. In this context Scalable Video Coding (SVC) techniques [1] allow multimedia terminals to accommodate the spatial and temporal resolutions and the quality of a decoded video sequence to the available resources (screen size, computational power or battery consumption).

SVC techniques have been defined in most video coding standards [2][3], but their use has not become widespread because of their jitter problems and poor efficiency [1]. However, the SVC features included in H.264 [4] surpasses those used in former standards and facilitate new possibilities.

Up to now, the available H.264/SVC decoder implementations are restricted to the PC domain. The JSVM [5] and Open SVC [6] are open source decoders while [7] is a proprietary implementation from the IMEC.

In this paper, a performance comparison between the two aforementioned open source H.264/SVC decoders is presented for both, PC and DSP environments. The performance has been measured in terms of time spent to decode each layer of a H.264/SVC stream.

The results obtained demonstrate that the Open SVC decoder is more suitable to be used in the implementation of embedded systems, because its performance is between five and twelve times better than achieved with the JSVM decoder.

The rest of the paper is organized as follows. The features of both decoders are outlined in section 2. In section 3, the process to migrate the decoders to the DSP environment is described. The results of the profiling tests are presented and discussed in section 4. Finally, section 5 concludes the paper.

2 H.264/SVC DECODERS PC-BASED IMPLEMENTATIONS

Currently, there are two H.264/SVC video decoder implementations with source code available: the JSVM decoder and the Open SVC decoder. In this section the main features of both implementations are summarized.

2.1 Joint Scalable Video Model (JSVM)

In this standard, the video compression is performed by generating a unique hierarchical stream with several layers with different resolution. As an example, Fig. 1 shows a diagram representing different scalability layers. Each box represents a coded Network Adaptation Layer (NAL) packet with a different spatial (DX), temporal (TX) and quality (QX) resolution. An SVC decoder may decode, e.g., only the shaded NALs to get full spatial resolution, half temporal resolution and a reasonable quality level. A different more powerful decoder might decode the entire stream to get full temporal and spatial resolution and higher quality.

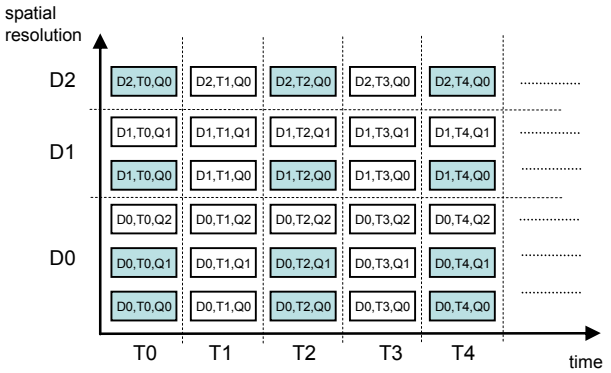

Fig. 1. Space-temporal diagram showing different levels of scalability

[1] This work was supported by the Spanish Ministry of Science and Innovation under grant TEC2009-14672-C02-01 (PccMuTe: Power Consumption Control in Multimedia Terminals).

As a part of the standardization effort, the Joint Scalable Video Model (JSVM) reference software [5] has been developed. Streams with different levels of scalability can be encoded and decoded with the testbench implemented in the JSVM. Table I shows some of the currently available C++ libraries used by the reference software.

TABLE I
SOME LIBRARIES USED IN THE JVSM CODE

Library	This library provides classes…
H264AVCCommon LibStatic (*Common*)	…used in both encoding and decoding process, e.g. to provide different data structures
H264AVCEncoder LibStatic (*Encoder*)	…used only in the encoder, e.g. to implement the motion estimation or the entropy coding
H264AVCDecoder LibStatic (*Decoder*)	…used only in the decoder, e.g. to implement the stream analysis or the entropy decoding
H264AVCVideoIo LibStatic (*VideoIo*)	…used to provide input video to the encoder or to store the decoded video (from/to files)
BitStreamExtractor	... used to extract in a file the selected layers from an other stream

The main advantages of this decoder are that it implements all the profiles and levels defined in the standard. However it has some important disadvantages for the implementation of embedded systems: the source code has been writing in C++, the performance has not been optimized and it is not possible to select the layer to be decoded and an additional tool, BitStreamExtractor, must be used previously to extract the layer to be decoded.

2.2 Open SVC Decoder

The Open SVC Decoder [6], a C language H.264/SVC Baseline Profile decoder, has been developed in the framework of Scalim@ges project and is improved within the SVC4QoE project [8]. This open source decoder is based on an AVC Baseline Profile decoder, and has been updated with most of tools of AVC Main profile and SVC Baseline Profile.

Open SVC Decoder is able to decode all type of scalability as temporal and quality scalability without any restrictions, and only 1.5 and 2 ratio for two consecutive enhancement layers for the spatial scalability as specified into the SVC Baseline Profile.

The Open SVC Decoder has been developed with the main idea to be deployed over different platforms with different operating system [9] such as Digital Signal Processor (DSP), Personal Digital Assistant (PDA), x86 or the Cell processor.

This decoder is able to decode a specific layer in the scalable structure of a stream. This particularity is very useful in a broadcast environment as the layer selection can be done during the decoding process. Moreover, when transmission errors occur, a part of the stream can be corrupted or missing; the decoder is then able to automatically switch to a lower layer, compensating thus transmission errors to optimize the visual quality of the video sequence.

One of the main advantages of this implementation is that the source code has been written in C language and the stream extraction functionality is integrated with the decoder. However, it does not support all the profiles and levels yet.

3 H.264/SVC DECODERS DSP-BASED IMPLEMENTATIONS

In this section, the process to migrate both decoders to the DSP environment is briefly introduced.

3.1 JSVM Migration Process

The JVSM reference software, designed to run on a PC environment, has been ported to the DSP. A separate project using the *Common*, *Decoder* and *VideoIo* libraries has been compiled and optimized.

The DSP development framework [10] supports C++ coding. Among other tasks, the porting process required the redefinition of some data types, the implementation of C functions not included in the DSP library (*fileno* and *fcntl*) and the rewriting of incomplete type definitions.

The source code of the *Common*, *Decoder* and *VideoIo* libraries has been adapted in order to get correct compilation and linking. Besides other minor changes made to the *Common* and *Decoder* libraries, non-aligned read operations have been replaced with DSP specific instructions and template specialization declarations have been added. In *VideoIo*, the code has been adapted to use the file management functions *fread*, *fwrite*, *fopen* and *fclose* instead of *read*, *write*, *open* and *close*.

The JSVM decoder always extracts the highest quality layer included in a stream. To decode a specific layer, first it must be extracted from the stream using the BitStreamExtractor library. This application generates a new file including only the selected layers. Obviously, this process can not be done in embedded applications with real time requirements.

As shown in Fig. 2, the stream extractor and the video decoder have been integrated in only one application using a share buffer between them. With this new application, the user selects the layer to be decoded using the DX, QX and TX parameters. The BitStreamExtractor module filters the NAL units associated to the selected layer in real time and stores them in a shared buffer. The decoder process the stream allocated in the shared buffer and save the uncompressed frames in a YUV file.

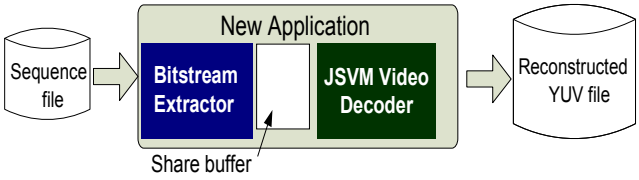

Fig. 2. Integration of BitstreamExtractor and Video Decoder using a shared buffer.

3.2 Open SVC Decoder Migration Process

The Open SVC decoder has been developed for a PC-based platform. The decoder has been ported to the DSP using the methodology presented in [11]. In the DSP-version, the decoder has been encapsulated into a DSP-BIOS task. Code and data have been allocated in external memory. The maximum size of the decoded pictures has been reduced from HD (1920×1080) to SD (720×576). The way

to select the layer to be decoded has been modified. In the original code, the layer was selected using the command line arguments while in the DSP version these parameters are introduced through a configuration file that is parsed at the beginning of the decoding process.
The conformance of the DSP-based decoder has been done using the sequences included in the standard [12].

4 RESULTS

4.1 Sequences Description

To test the decoders six well-known video streams (Akiyo, Coastguard, Flower, Foreman, Mobile and News) have been encoded using a commercial H.264/SVC encoder [13]. Two different sets of test sequences have been generated to evaluate the influence of a specific layer embedded on the video stream in the decoder performance. For each type of set, sequences that consist of six layers extracted out from the eight possible combinations among two spatial resolutions (QCIF and CIF), two frame-rates (12.5 and 25 frames per second) and two qualities (low and high) have been generated. Furthermore, for each sequence the total bitrate and that of the base layer are 512 Kbps and 102 Kbps (20% of a total bitrate of 512 Kbps), respectively.
The stream structure of the first set of test sequences, exemplified with the *Akiyo* sequence, can be seen in Fig. 3. Note that the two possible temporal scalability values are omitted. In this type of test sequences, the first enhancement layers are derived from the corresponding base layers with only an increase in quality while the second enhancement layers are derived from the previous ones with only an increase in spatial resolution. In this paper, they are designated as *quality-spatial* sequences to stress the fact that the greatest quality layer is obtained from the base layer with, first, a quality improvement and, then, with a spatial resolution improvement.

Fig. 3. Quality-Spatial six-layered test sequence structure.

Fig. 4 shows the stream structure of the second set of test sequences. The first enhancement layers are derived from the base layers with only an increase in spatial resolution although the second enhancement layers are generated from the first enhancement ones with an increase in quality. In the rest of the paper the sequences belonging to this set are designated as *spatial-quality* sequence.
As far as the codec parameters to generate the test sequences concern, the GOP size equals 8 frames, the CABAC is used for entropy coding, the deblocking filter is active, all possible macroblock partitions are enabled for inter-prediction, three reference frames are allowed, and one B-frame is coded for each I-frame. All the generated sequences have 880 frames.

Fig. 4. Spatial-Quality six-layered test sequence structure.

4.2 PC-Based Decoder Performance

In this subsection the performance of the H.264/SVC decoders is presented for a PC environment. The test-bench used to evaluate the PC performance is based on a dual-core processor running at 3GHz with 3 GB of RAM memory.
Both decoders have been modified to measure the average time used to decode a complete stream using PC internal timers.
The two set of sequences described in section 4.1 have been decoded with both decoders. The time spent by the decoders for each layer of each sequence has been measured. Then, the results has been averaged over the set of spatial-quality and quality-spatial sequences.
In Table II, the average time in milliseconds spent to decode all the layers of all the quality-spatial sequences is presented for both decoders.
The number of frames decoded for each layer is different depending of the temporal scalability. For the layers with a temporal resolution of 25 frames per second, 880 frames have been decoded; however, for the layers with half temporal resolution, the number of decoded frames is 440.
Moreover, the Open SVC speed-up with regard to the JSVM implementation, i.e. the quotient between the average Open SVC rate and that of the JSVM is included for each layer.

TABLE II
PERFORMANCE COMPARATION BETWEEN JSVM AND OPEN SVC DECODER USING A PC FOR QUALITY-SPATIAL SEQUENCES (IN MSEC)

	QCIF 12.5 fps Low	**QCIF 25 fps Low**	**QCIF 12.5 fps High**	**QCIF 25 fps High**	**CIF 12.5 fps High**	**CIF 25 fps High**
JSVM	1178	2040	1992	3451	12378	22938
Open SVC	293	496	613	1207	1727	2939
Speed-up	4.3	4.4	8.2	8.6	6.0	6.5

Table III presents the same information that Table II but for the quality-spatial sequences.

TABLE III
PERFORMANCE COMPARATION BETWEEN JSVM AND OPEN SVC DECODER USING A PC FOR SPATIAL-QUALITY SEQUENCES (IN MSEC)

	QCIF 12.5 fps Low	QCIF 25 fps Low	CIF 12.5 fps Low	CIF 25 fps Low	CIF 12.5 fps High	CIF 25 fps High
JSVM	1164	1949	11017	20983	14843	27980
Open SVC	272	444	1341	2439	2466	4297
Speed-up	4.0	4.1	3.2	2.9	7.2	7.8

The conclusions obtained analyzing the results presented in Table II and Table III are the following:

- Both decoders achieve real-time performance for all layers.
- The average speed-up remains almost unchanged for frame rate variations.
- The performance of decoding a base layer with Open SVC decoder is approximately four times greater than the performance when the JSVM is employed.
- The ratio between the Open SVC average speed-up of the layers that experiment either a quality or spatial improvement remains invariant for spatial-quality and quality-spatial sequences.

4.3 DSP-based Decoder Performance

In this subsection, the test bench proposed to measure the decoder performance in the DSP-based environment is described. The results obtained for JSVM and Open SVC decoders are showed.

a) Test bench used to measure the performance

A block diagram of the test-bench is shown in Fig. 5. As can be seen, first, a test stream is read from a file and written into a stream buffer allocated in external memory. Then, the decoder reads the stream from the memory and decodes it on a picture basis. At last, the decoded picture is written into a buffer and also into a component YUV video file.

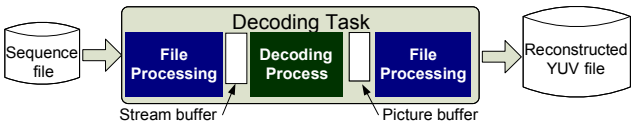

Fig. 5. Test-bench block diagram to profile the decoders.

A fixed-point video-oriented DSP [14] was used to implement the H.264/SVC decoders. In Fig. 6, a simplified block diagram of the DSP internal architecture is shown.

The DSP is based on high-performance VLIW architecture. Two levels of internal memory (L1 and L2) are available. The L1P memory/cache consists of a 32 KB memory space that can be configured as general purpose mapped memory, direct mapped cache or combinations of the two. The L1D memory consists of an 80 KB memory space that can be entirely configured as general purpose memory. Instead, up to 32 KB of L1D can be configured as a 2-way set-associative cache. Finally, the L2 memory/cache consists of a 128 KB memory space, shared between program and data. L2 memory can be configured as a general purpose mapped memory, a cache memory, or a combination of both.

For the Open SVC implementation, the internal memory has been configured as follows: L1D is divided in 32 KB for cache memory and 48 KB for general purpose data; L1P is configured as a 32 KB cache program memory and L2 is splitted between level-2 cache memory and general purpose memory.

A switched central resource interconnects the core with a set of standard peripherals and a video processing subsystem. The external memory is accessed through a dedicated interface, EMIF, using a 64-bit data interface. The other peripherals are an EDMA controller (EDMA2), two video ports, an Ethernet port (EMAC), an output audio interface (McASP) and several general-purpose I/O pins (GPIO).

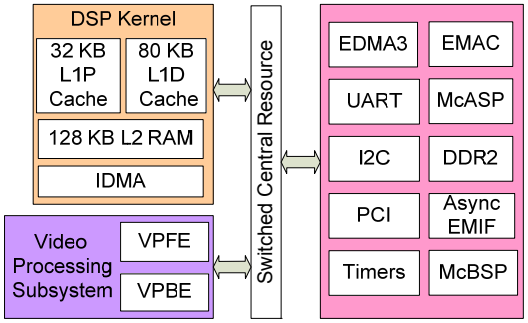

Fig. 6 Architecture of the DSP.

A commercial prototyping board [15] based on this DSP has been used to measure the decoder performance (see Fig. 7). The board includes a DSP working at 594 MHz, 128 MB of SDRAM external memory, 80 MB of Flash external memory and several interfaces.

Fig. 7. The DSP based development board.

b) Performance Results

The two set of sequences described in section 4.1 have been decoded with the DSP-based versions of the JSVM and Open SVC decoders.

The time spent by the DSP to decode each layer of each sequence has been measured using DSP internal timers. In this measure, the time used to access the files has not been included.

The number of frames decoded depends of the temporal scalability of each layer. For 25 frames per second temporal resolution layers, 880 frames have been decoded; while for layers with half temporal resolution 440 frames are decoded.
The Table IV shows the average time in seconds spent by the DSP when it decodes each layer included in the quality-spatial sequences using both decoders. In the last row, the Open SVC decoder speed-up with regard to the JSVM implementation is included for each layer.

TABLE IV
JSVM AND OPEN SVC DECODER PERFORMANCE FOR QUALITY-SPATIAL SEQUENCES (IN SEC).

	QCIF 12.5 fps Low	**QCIF 25 fps Low**	**QCIF 12.5 fps High**	**QCIF 25 fps High**	**CIF 12.5 fps High**	**CIF 25 fps High**
JSVM	23.5	45.0	40.2	74.6	204.6	403.2
Open SVC	3.3	6.3	7.1	14.9	18.2	37.4
Speed-up	7.2	7.2	5.6	5.0	11.2	10.8

Table V presents the same information that Table IV but for the quality-spatial sequences.

TABLE V
JSVM AND OPEN SVC DECODER PERFORMANCE FOR SPATIAL-QUALITY SEQUENCES (IN SEC).

	QCIF 12.5 fps Low	**QCIF 25 fps Low**	**CIF 12.5 fps Low**	**CIF 25 fps Low**	**CIF 12.5 fps High**	**CIF 25 fps High**
JSVM	22.8	44.0	196.3	375.3	242.7	473.2
Open SVC	3.2	6.3	14.7	30.9	27.2	58.5
Speed-up	7.1	7.0	13.3	12.1	8.9	8.1

The conclusions obtained analyzing the results presented in Table IV and Table V for the DSP-based decoders are similar than those derived for PC-based. Some additional conclusions are the following:

- The speed-up achieved in the DSP environment for each layer of all the sequences is greater that obtained for the PC.
- The JSVM decoder achieves no real time performance even for the base layers.
- The Open SVC decoder achieves real time performance but for the highest quality layer (CIF, 25 fps and High) of both set of sequences.

As result of this analysis the Open SVC has been selected as starting point for the implementation of a mobile terminal device. This decoder achieves no real time performance with some layers of the generated streams but is not so far of this objective. For the set of quality-spatial sequences 23.5 frames per second are decoded, while for the set of spatial-quality sequences 15.0 frames per second are decoded.

The methodologies presented in [16][17] are been applied to reduce the time spent to decode the H.264/SVC sequences. These methodologies improve the decoder performance taking advantage of the SIMD (Simple Instruction Multiple Data) architecture, using explicit DMA transfers to move data between internal and external memory and allocating code and data in the different levels of internal memory to reduce the cache misses (the first results obtained after the optimization process can be checked at [18]).

5 CONCLUSIONS

In this paper, a performance comparison of two H.264/SVC-compliant video decoders, JSVM and Open SVC, is presented to select the decoder to implement on a DSP-based multimedia mobile terminal. The decoders were initially developed for PC environment so a migration process to a DSP-based environment is needed.
The comparison shows that the Open SVC decoder clearly outperforms the JSVM implementation for the PC- and DSP-based environments.
As far as the frame rate variation concern the average speed-up for both environments is not significantly affected. In addition, the PC performance of decoding a base layer with Open SVC is approximately four times greater than the performance when the JSVM is employed. In contrast, for DSP the performance is seven times greater.
For DSP environment, the JSVM decoder achieves no real time performance even for the base layers. However, the Open SVC decoder achieves real time performance except for the highest quality layer (CIF, 25 fps and High). Therefore, the application of time-based optimization methodologies to achieve real time performance is only worthy for Open SVC decoder.

6 ACKNOWLEDGMENT

The authors would like to thank Ernesto Seisdedos, David Samper and Alejandro González from Grupo de Diseño Electrónico y Microelectrónico (UPM) for their contributions to this work.

7 REFERENCES

[1] J-R Ohm, "Advances in Scalable Video Coding". *Proceedings of the IEEE*, vol. 93, nº 1 pp. 42-56, Jan. 2005.
[2] ISO/IEC 13818-2 (ITU-T Rec. H.262). Generic coding of moving pictures and associated audio information: Video. 1995.
[3] ISO/IEC 14496-2. Information technology. Coding of audio visual objects. Part 2: Video. 1998.
[4] Joint Draft ITU-T Rec. H.264 | ISO/IEC 14496-10 / Amd.3 Scalable Video Coding, July 2007.
[5] Joint Scalable Video Model JSVM-9.9, Available in CVS repository at Rheinisch-Westfälische Technische Hochschule (RWTH) Aachen.
[6] M. Blestel and M. Raulet, "Open SVC Decoder", http://opensvcdecoder.sourceforge.net/.
[7] IMEC. http://www2.imec.be/be_en/press/imec-news/archive-2008/imec-speeds-up-scalable-video-decoder-svc-with-factor-20.html
[8] M. Barkowsy, M. Blestel et al, "Overview of the SVC4QoE project", 6th Int. ICST Conf. on Mobile Multimedia Communications Sept 2010.
[9] M. Pelcat, M. Blestel and M. Raulet. "From AVC decoder to SVC: Minor impact on a data flow graph description" PCS2007. June 2007.

[10] Texas Instruments Incorporated. “Code Composer Studio v4.0 IDE Getting Started Guide”
[11] F. Pescador, D. Samper, M.J. Garrido, E. Juárez and M. Blestel. "A DSP based SVC IP STB using Open SVC Decoder". Int. Symposium on Consumer Electronics. Braunschweig Germany, 7-10 June 2010.
[12] Sequences available at http://wftp3.itu.int/av-arch/jvt-site /draft_conformance/
[13] MainConcept. SVC Baseline SDK DirectShow. Version 1.0.0. Aachen, Septembre 22, 2009.
[14] Texas Instruments. DaVinci DSPs. http://focus.ti.com/docs/prod/ folders/ print/tms320dm6437.html.
[15] DM6437 Digital Video Development Platform (DVDP). http://www.spectrumdigital.com/product_info.php?cPath=37&products_id=196&osCsid=0abf0072f9687529d1d010374287bd64.
[16] F. Pescador, G. Maturana, M.J. Garrido, E. Juárez and C. Sanz “An H.264 video decoder based on a DM6437 DSP”. IEEE Trans on Consumer Electronics. Vol. 55, Nº 1. Pp. 205-212. February 2009.
[17] F. Pescador C. Sanz, M.J. Garrido, E. Juarez y D.Samper. “A DSP Based H.264 Decoder for a Multi-Format IP Set-Top Box”. IEEE Trans. on Consumer Electronics. Vol 54. February 2008.
[18] F. Pescador, D. Samper, E. Juárez, M. Raulet and C. Sanz. “A DSP Based H.264/SVC Decoder for a Multimedia Terminal”. IEEE International Conference on Consumer Electronics. Las Vegas EEUU, January 2011.
[19]