TOWARD FREE AND OPEN SOURCE FILM PROJECTION FOR DIGITAL CINEMA

Nicolas Bertrand
Jean-Denis Durou
Vincent Charvillat
IRIT, UMR CNRS 505
Toulouse, France
E-mail: nicolas.bertrand@isf.cc

Carsten Griwodz
Simula Research Laboratory
Oslo, Norway
E-mail: griff@simula.no

KEYWORDS
Image Compression, Digital Cinema, JPEG2000, Open Source, VLC, libavcodec.

ABSTRACT
Cinema industry has chosen Digital Cinema Package (DCP) as encoding format for the distribution of digital films. DCP uses JPEG2000 for video compression. An efficient implementation of coding and decoding for this format is complex, however. Currently deployed equipment is expensive and has high maintenance costs, preventing art-house cinema theaters from acquiring it. Therefore, we conduct this research activity in cooperation with Utopia cinemas, a group of art-house cinemas, whose main requirement (besides functional ones) is to provide Free and Open Source Software (FOSS). This paper presents a solution that achieves real-time JPEG2000 decoding and DCP presentation based on widespread open source multimedia tools, namely VLC and libavcodec library. We present the improvements that were made in VLC to support the DCP packaging format, as well as details on JPEG2000 decoding inside libavcodec (optimization and lossy decoding). We also evaluate the performance of the decoding chain.

I. INTRODUCTION

Cinema theaters switched from 35 mm prints to digital era. The Digital Cinema System Specification (DCSS) [1], provided by the Digital Cinema Initiative (DCI), is now a world-wide standard. The specification describes how to create, distribute and project a Digital Cinema Package (DCP). It requires JPEG2000 as intra-frame codec for video, WAV for audio, and XML for subtitles. Image dimensions are specified for 2K and 4K, with three 12-bit color components sequenced with a frequency of at least 24 fps (frames per second). The DCP size is typically about 80–200 GB according to film duration or compression ratio.

Our research is conducted in collaboration with Utopia cinemas (five independent theaters in France). Those theaters initiated this project because they need to understand the implications of changing to digital. Primarily, they are concerned about becoming dependent on a single company's technology for presentation, and want us to provide Free and Open Source Software (FOSS) for decoding films distributed in the DCP format.

The JPEG2000 [2] format was selected by DCI for video compression because of its compression efficiency. The implementation of a fast JPEG2000 encoder and decoder, however, is complex. Commercial equipment currently used by cinemas relies on VLSI hardware for decompressing DCP at the required frame rate. This kind of equipment is expensive, and not affordable for art-house cinema theaters. Our research goal is to lower the cost for DCP playback by developing a software that runs on today's standard off-the-shelf hardware. There are already software solutions, such as Kakadu [3] and EasyDCP [4] for real-time JPEG2000 decompression, but these are not FOSS solutions. Kakadu is a JPEG2000 coder/decoder that supports all kinds of JPEG2000 profiles, but cannot playback DCP. EasyDCP is dedicated to DCP playback and can play DCP in real-time, but as Kakadu, the software is not FOSS.

Our selection of VLC (Video LAN Client) as the basis for our DCP decoder is due to its position as a FOSS solution with high flexibility, performance and proliferation. We have implemented a DCP module inside VLC, and a JPEG2000 decoder inside the libavcodec multimedia library (used by VLC).

In the next section, we review current fast software implementations for JPEG2000 decoding. Then, we present our projection system called OpenSMS (Screen Management System) in Section III. In Section IV we present the VLC architecture, and the design of our module. The JPEG2000 decoder (named J2K-libavcodec) implementation is detailed in Section V. In Section VI, validation and performance measurements of our solution are presented.

II. STATE OF THE ART – THE PROBLEM

The key point for DCP playback is to reach a frame rate of 24 fps with synchronized playout of audio, video and subtitles. The bottleneck for achieving this is the computational complexity of JPEG2000 decoding. Overcoming it in software requires parallel computation, and there are currently three dominant approaches to this: the use of multi-threading and processor-specific multimedia extensions like Streaming SIMD Extensions (SSE), the use of GPGPU, or a combination thereof. For example, Taubman [5] relies on multi-threading and SSE, the patented EasyDCP [6] relies on GPGPU to present a GOP (Group Of Pictures) approach based on similarity between codeblocks to avoid latency, Le [7] mixes both approaches. Le [8] and Matela [9] explain proposals for parallelizing EBCOT using GPGPU. Taubman [5] shows

that a decompression of 24 fps can be achieved with multi-threading and SSE instructions. This solution is implemented in Kakadu. Avoiding GPGPU reduces the decoder's complexity and hardware dependence; the dependency on NVIDIA hardware due to the use of CUDA is a limitation of solutions by Le [8] and Matela [9]. Although OpenCL [7] is meant to be platform-agnostic, detailed hardware-specific tuning is required to achieve a high performance.

Another way to reduce the decompression time is to not completely decode all the compressed bitstream. In Jimenez [10], a method for visually lossless decompression is presented. The method is based on a perceptual model designed by the authors for compression. The proposed strategy is as follows for each codeblock: extract the most significant bitplanes from the codeblock header, compute a Visibility Threshold (VT), based on variance estimation and the author's perceptual model. Then, during codeblock's decompression, at each coding pass an error is compared to VT. If the error is lower than VT, remaining coding passes are skipped. Kakadu also proposes an option called "bitstream truncation" to achieve better decompression performance by stripping away coding passes.

We decided to implement a decoding solution that combines multi-threading, SSE instructions and basic skip of coding passes for least significant bitplanes codeblock's coefficients. This decoding technique is a key component of our alternative projection system.

III. PROJECTION SYSTEM

The standard DCI projection system and our OpenSMS are juxtaposed in Figure 1. Our goal is not to propose a new projection system, but to provide a simpler version of an existing one. Table I presents the features of both systems.

A DCI projection system is composed by a storage device (where the DCPs are ingested), connected to an Integrated Media Block (IMB) via a PCI-Express link. The IMB is in charge of media decryption and decompression. Plain decoded images are sent to the projector, the sound processor receives the plain audio channels.

TABLE I: DCI Projection System Features Compared to OpenSMS.

Feature	DCI	OpenSMS
Picture Size	2K	2K interpolated
DCP 4K Support	Yes	Yes †
Picture Depth	12 bits	10 bits
Sound	PCM 24 bits	PCM 24 bits
Subtitle	Yes	Yes
DCP Decryption	Yes	Yes
Security Manager	Yes	No
Physical Security	Yes	No
Secure Logging	Yes	No
Forensic Watermarking	Yes	No

†To Be Implemented

Our OpenSMS system is composed by an off-the-shelf hardware and an "e-cinema" projector (Barco RLMW8, using tri-DLP as DCI projectors). For our system validation, we have defined two hardwares: a laptop with a quad core Intel i7 and for better performance, a bi-Xeon with 12 cores. The hardware and the projector are connected via a HDMI link, using HDCP encryption.

We have also decided to not implement all the security requirements described in specification [1]. The main reason is to reduce the system complexity and we think that the security constraints do not fit with the projection policy of art-house cinema. Nevertheless we are aware of content protection, and plan to implement playback of encrypted DCP.

Our OpenSMS software is based on VLC and is presented in Section IV.

After video decompression, to preserve correct color display in a monitor or a projector, we implemented a GLSL filter to convert from CIE 1931 XYZ to the required color space, usually sRGB.

IV. DCP PLAYBACK IN VLC

VLC is a widespread cross-platform FOSS media player, downloaded more than 1.4 billion times. VLC is essentially a multimedia framework, where you can dynamically load modules according to the input (files, network streams) and the outputs (audio or video, on screen or network). The framework's core handles low-level operations like threading, timing and synchronization. It is also in charge of pipelining the media streams from input to output by connecting modules, used to do the media processing work. An example of modules loaded for the DCP case is illustrated in Figure 2. Each module has a type according to its purpose, like access, demux, codec, video_output, ...

VLC depends on a large number of FOSS libraries including libavcodec.

Our first contribution is the creation of a DCP module, VLC had not handled it so far. JPEG2000 decoding is done by our decoder (J2K-libavcodec) implemented in libavcodec (cf. Section V). The OpenGL module was slightly modified to implement in GLSL the XYZ to sRGB conversion.

Next we present our module design. A DCP is a set of files stored in a folder. DCP meta-data are stored in XML files, while MXF containers are used to store the media *essence* (video, audio, subtitles). There is at least one MXF file per essence, but several audio and subtitles MXF files can be stored in the folder to handle several languages.

A DCP is a container (the folder) that in turn contains several other containers (the MXF files). Operations for *access* and *demux* are not easily separated, so an *access_demux* module was judged more suitable for DCP. The essence files are accessed via *asdcplib* external library. We chose this library because it supports MXF containers and DCP essence types, and it is popular in cinema tools. Once the essences are extracted, VLC Elementary Streams (ES) are created, and

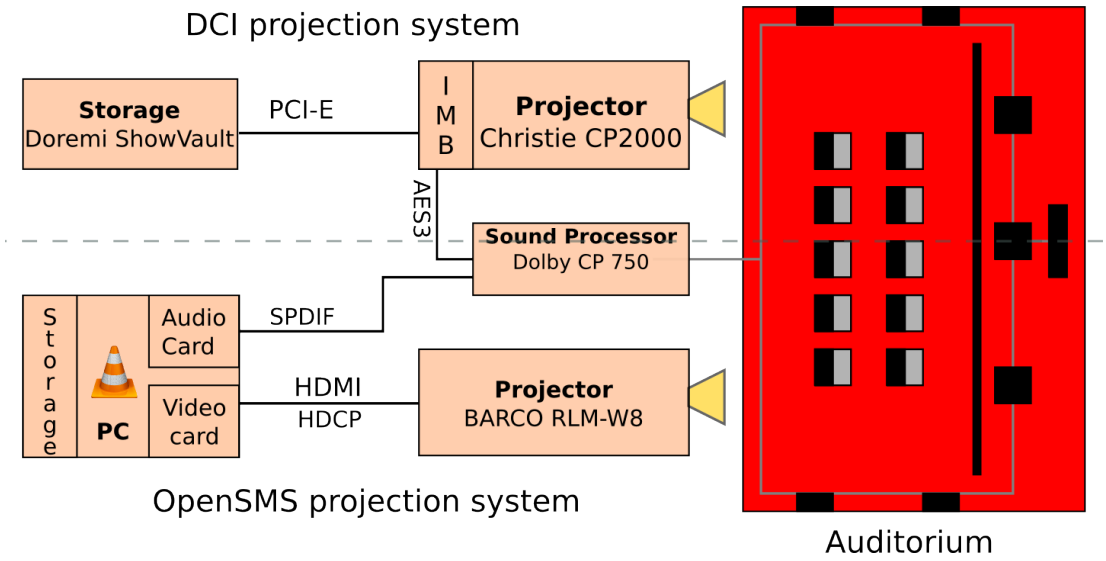

Fig. 1: Digital cinema Projection Systems. The DCI media block, integrated in the projector, performs decoding. Our system (OpenSMS) is not integrated, and can be connected to all kinds of projectors, or display directly on a standard monitor.

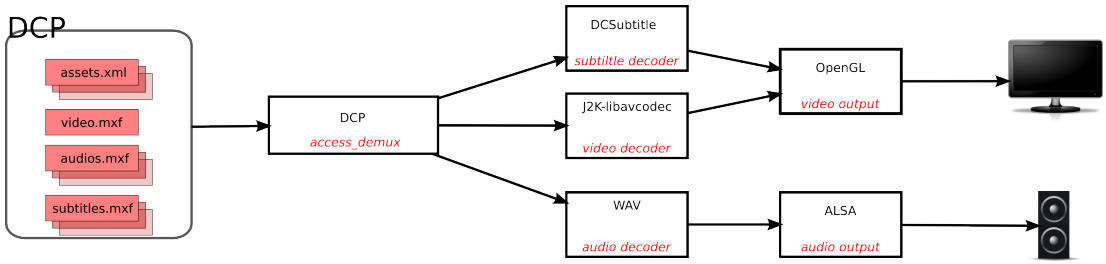

Fig. 2: DCP Playback in VLC and libavcodec. Each box represents a VLC module, and its functionality is written in red. The arrows represent the streams between modules.

sent to the decoders. ES, one per media, form the interface between the demuxers and the decoders.

VLC design is highly multi-threaded, modules are executed in separate threads. Consequently, demultiplexing, decoding and display are executed asynchronously. The synchronization of audio, video and subtitle is performed by a dating mechanism called *Presentation Time Stamps*. The date is set in the ES, through our module and the output modules use this date to play at the right time.

Our DCP module is publicly available in VLC master branch since December 2013.

Code for DCP VLC module is available in git.videolan.org, stored in directory modules/access/dcp (http://git.videolan.org/?p=vlc.git;a=tree;f=modules/access/dcp).

Even if the acceptance of the module required a painstaking work, it was a big step towards a FOSS DCP playback solution.

V. JPEG2000 DECODER IMPLEMENTATION

Our second contribution is a specific JPEG2000 decoder for libavcodec, a coder/decoder multimedia library. Like VLC, libavcodec is a FOSS and cross-platform project. Our JPEG2000 decoder is not the only one. Libavcodec can also use OpenJPEG library for decoding JPEG2000 files. OpenJPEG is a FOSS library hat accepts all JPEG2000 profiles, but is mostly aimed at coding still images. In contrast to this, our codec is aimed at decoding JPEG2000 videos, especially for cinema. By specializing, we can achieve smaller structures and simpler code, avoiding OpenJPEG's complex structure that is required to support all JPEG2000 options.

As the decoder is intra-frame, we decided to multi-thread the decoder at frame level with threading mechanisms provided by libavcodec primitives.

The inverse Discrete Wavelet Transform (DWT) and Main Component Transform (MCT) was optimized with SSE instruction set. The SSE optimization is widely inspired from OpenJPEG, and by focusing on structure size.

The lifting based 2D DWT is optimized by SSE instructions. The six steps of irreversible 1D filtering (cf. chapter F.2.8.2 in JPEG2000 standard [2]) are executed through two SSE functions. One function for the two first steps (linear computes), the other for the remaining steps (nonlinear computes). Implementing SSE instructions for MCT is easy, since it is a matrix multiplication.

Furthermore, for decoding, we increase the performance by

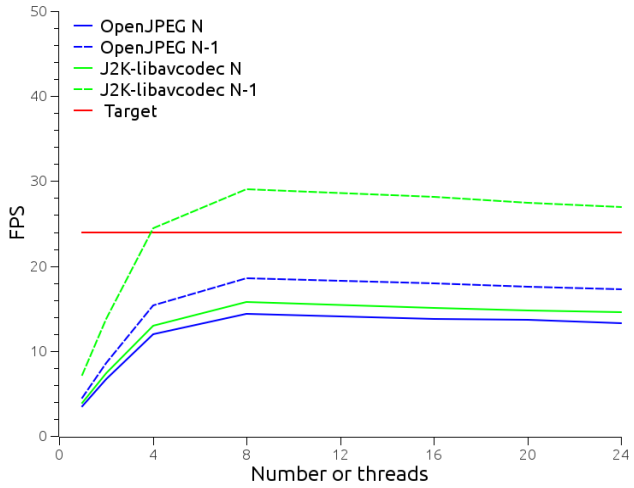

Fig. 3: JPEG2000 Decoders Performance on Machine 1 ($N = 5$ resolution levels).

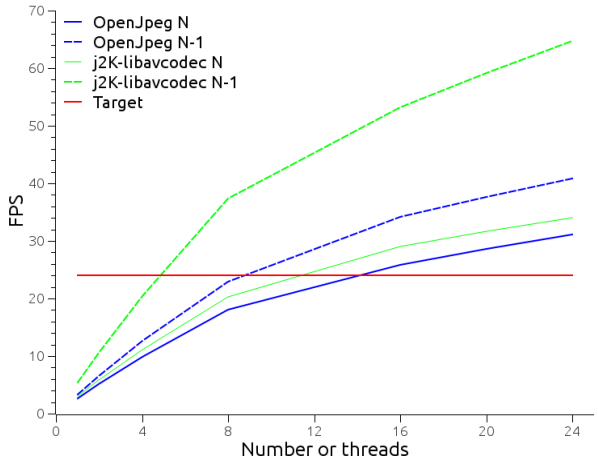

Fig. 4: JPEG2000 Decoders Performances on Machine 2 ($N = 5$ resolution levels).

doing only once some parsing of JPEG2000 headers, as the digital cinema profiles force many parameters.

Our J2K-libavcodec is also publicly available in libavcodec. To get the released code clone git://git.libav.org/libav.git, the JPEG2000 decoder is stored in libavcodec directory, the decoder is stored in jpeg2000* files. All the optimizations are not yet pushed in libavcodec, but are publicly available in our Gitorious account (clone git@gitorious.org:libav/nicoisfs-dondiego-libav.git and checkout expev2 branch for JPEG2000 decoders with all optimizations).

The decoder evaluation and comparison with OpenJPEG is presented in next section.

VI. EXPERIMENTAL VALIDATION

A. JPEG2000 Evaluation on libavcodec

The tests are made on two machines. Machine 1 is a laptop with an Intel Core i7-2820QM operating at 2.30 GHz, the processor has 4 cores for a total of 8 SMT threads (hyper-threads in Intel terminology). Machine 2 is a bi-Xeon E5-2620 operating at 2.0 GHz. Each chip has 6 physical cores, resulting in a total of 24 SMT threads. All tests are conducted with hyperthreading turned on. The tests are performed according to the number of CPU-threads and resolution levels. The tested DCP is the trailer of Moonrise Kingdom, and all numbers refer to decompression of the first 10 seconds (240 frames). The movie was encoded with a bit rate of 128 Mbits/s. Moonrise Kingdom image size is 1998×1080 with 5 resolution levels. The number N of resolution levels tested varies between 5 (2K format, 1998×1080) and 4 (1K format, 999×540).

In Figure 3, the comparison of J2K-libavcodec with OpenJPEG is presented (Machine 1). As the CPU has 8 SMT threads, the decoding performance with both decoders is limited to 8 libavcodec threads. A performance decrease has been observed if we use more libavcodec threads than the CPU has virtual cores. In 2K neither OpenJPEG nor J2K-libavcodec reaches the target (red line) of 24 fps; at least our decoder performs better. In 1K J2K-libavcodec reaches the target with 4 threads.

For Machine 2, the results are presented in Figure 4. All decoders, regardless the resolution level, reach the target of 24 fps: J2K-libavcodec reaches the target frame rate with 12 threads, OpenJPEG with 16 threads.

The fps gain of our decoder at 1K is high. There are two reasons. First, the image to decode is smaller (4 times smaller than 2K), so there are less codeblocks to decode. We cannot expect a gain close to 4 as the non-decoded codeblocks are the high frequency ones: EBCOT encodes high frequency codeblocks with less bits than low frequency. The other reason is memory usage and CPU cache management. With OpenJPEG, we achieve a gain of 1.5 between 2K and 1K, while ours is around 2. The main reason for this difference is memory management: we have smaller structures for decoding data flow, resulting in a reduction of cache misses and page faults.

TABLE II: Fps Comparison of Movies at Several Compression Bit Rates. Run on Machine 2 with J2K-libavcodec in 2K.

Movie	Bit rate (Mbits/s)	12 threads (fps)	24 threads (fps)
Spring Breakers	201	19.05	24.62
Django Unchained	151	19.58	24.64
Moonrise Kingdom	128	27.12	33.76
Jeux d'été	79	30.34	38.28

Table II shows the evolution of the fps according to the compression bit rate. The selected movies are DCP trailers shown in movie theaters. The decoding is tested on the first 240 frames. As expected, lower compression bit rates lead to higher frame rates. For Spring Breakers and Django Unchained we have almost the same fps. Spring Breakers has many black images and completely decoding black images is fast for a JPEG2000 decoder.

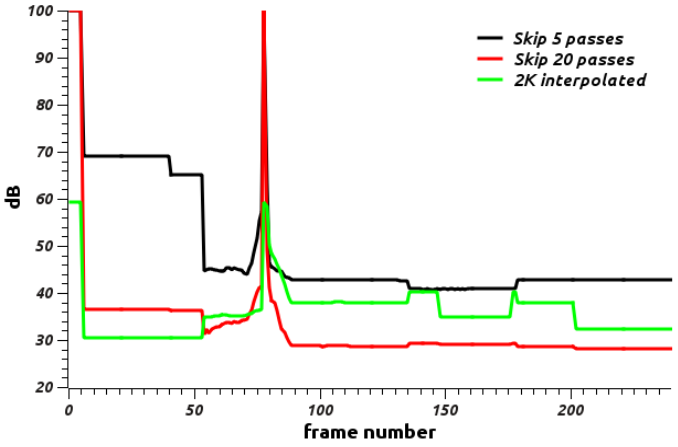

Fig. 5: PSNR Comparison. A peak is present around frame 80; this is due to the display of incrusted subtitles in our sample.

B. Image Quality Measurements

We present here tests of video quality measurements. The used sample is the same as the one used for performance measurements. The reference is the full 2K sample of Moonrise Kingdom. To test performance of lossy decoding, we skip the last decoding passes of EBCOT, by this way we skip the least significant bits of codeblocks coefficients. In Figure 5 we present the PSNR for the 240 frames of the sample. The measures are made by skipping 5 and 20 passes, and also the PSNR of the J2K-libavcodec with decompression at 1K and a bicubic interpolation to reach 2K.

By visual inspection of the video with 5 passes skipped, the image quality is acceptable for projection. With 20 passes skipped the image quality is not acceptable. The 240 frames are decomposed as follows: black image, 2 logos, movie sequence, black image, and 3 movie sequences. The variation of PSNR shows the various decomposition of the video. For 2K interpolated during the same sequences, variations of PSNR are observed. This is due to the incrustation of subtitles in the input video. When PSNR goes down subtitles appear.

TABLE III: Bit Rate (in fps) at Several Resolution Levels and Several Pass Skips for Moonrise Kingdom Sample.

	N	$N-1$
Skip pass 0	33.21	63.16
Skip pass 5	46.08	94.71
Skip pass 20	61.26	204.08

Table III presents fps performance in pass skipping. The test was executed on Machine 2. By skipping 5 passes, for N and $N-1$ resolution levels, a great performance gain is achieved (respectively 40% and 50%). More gain is achieved by skipping 20 passes, but the image quality is not acceptable. As stated by Jimenez [10] for image quality measurement, we confirm that visually lossless decoding is also computationally advantageous for JPEG2000 cinema streams.

C. DCP Playback Evaluation

The full read of DCP with audio and video synchronized, and correct color space display is made at VLC level. There are no tools inside VLC to measure the global performance of full decoding, only warnings or errors can be raised, indicating for instance loss of synchronization between video and audio, or decoding that is too slow. We used those warnings and visual inspection of DCP to evaluate.

TABLE IV: DCP Read at 24 fps, at Several Resolution Levels. 'Yes' indicates that DCP is read without desynchronization.

Machine	2K	2K interpolated (from 1K)
1 (i7 / 4 cores)	No	Yes
2 (bi-Xeon / 12 cores)	No	Yes

The evaluation results are reported in Table IV. We do not succeed to play the DCP at full scale, even with the decoding performances presented before. This is due to remaining work on our VLC module (pre-caching of video and limitation to 16 threads for decoding the video). We can play in interpolated 2K format. The codec provides 1K streams to VLC, the interpolation is made by VLC in the *video_output* module. The execution of the interpolation has no impact in DCP playback (no image freeze, no audio/video desynchronization). The samples used for those evaluations are available in http://utopialab.tetaneutral.net/DCPsamples/ website.

VII. CONCLUSION AND PERSPECTIVES

We have compared our decoder performance with OpenJPEG, the reference and fastest FOSS decoder. We have shown that our decoder is faster, and outperforms OpenJPEG when the highest frequency resolution level is skipped. Skipping some LSB decoding passes in EBCOT provides a great improvement on performances.

Our future work on the decoding part will consist in implementing multi-threading at codeblock level and further reducing data structure size. The purpose of that is to have piece of code that depends only on local data (i.e. no reference to higher level data). In this way, data structure should be kept in L1 and L2 caches during the execution and reduce the data time access.

We need also to validate the pass skipping and the 1K to 2K interpolation at system level to achieve that we will organize visual perceptions tests in cinema projection room. The tests will compare our system with current DCI projection system.

A comparison with closed-source software is more cumbersome and left for future work. Some performance results for Kakadu are presented on the official website, but a direct comparison is impossible because the sample sequences are not provided. The sample image size is 20% smaller, but the compression bit rate (244 Mbits/s) is twice higher than the one from our test sequence Moonrise Kingdom. On a bi-Xeon, Kakadu achieves a framerate of 24.24 fps (35.08 fps with the

separate ”speed pack”). In comparison, we achieve 34.04 fps on the same kind of machine. Results for EasyDCP are not publicly available.

As the number of cores increases in current architectures, we expect to avoid the usage of GPGPU at decoding level. In the overall pipeline however, we will use GPGPU, in particular for filtering (color space change and/or image color corrections) after decoding.

At VLC level, we need to optimize the DCP module to handle audio, video and subtitle synchronization with video in full resolution. Asdcplib allows management of encrypted MXF files, so we will implement handling of encrypted DCP. This feature will also be very useful for cinema exhibitors.

We are not far, from the point of view of performance and functionality, from a complete FOSS solution for DCP playback, usable by movie distributors, to preview the DCP, and by exhibitors for DCP playback.

REFERENCES

[1] DCI, *Digital Cinema System Specification*, Digital Cinema Initiatives, 1.2 with errata edition, Mar. 2012.

[2] ITU-T, *Information technology – JPEG2000 image coding system – Part 1: Core coding system*, Aug. 2002.

[3] UNSW, “Kakadu software,” July 2013.

[4] FraunHoffer IIS, “EasyDCP software suite,” July 2013.

[5] D. Taubman, “Multithreaded processing paradigms for JPEG2000,” in *MMSP 2012*, 2012, pp. 164–169.

[6] V. Bruns, H. Sparenberg, and S. Fossel, “Video decoder and methods for decoding a sequence of pictures, patent wo 2012004164 a1,” Jan. 2012.

[7] R. Le, J.L. Mundy, and R.I. Bahar, “High Performance Parallel JPEG2000 Streaming Decoder Using GPGPU-CPU Heterogeneous System,” in *ASAP 2012*, 2012, pp. 16–23.

[8] R. Le, I.R. Bahar, and J.L. Mundy, “A novel parallel Tier-1 coder for JPEG2000 using GPUs,” in *SASP 2011*, 2011, pp. 129–136.

[9] J. Matela, M. Šrom, and P. Holub, “Low GPU Occupancy Approach to Fast Arithmetic Coding in JPEG2000,” in *LNCS 7119*, pp. 136–145. 2012.

[10] L. Jimenez-Rodriguez, F. Auli-Llinas, and M.W. Marcellin, “Visually lossless strategies to decode and transmit JPEG2000 imagery,” *Signal Processing Letters, IEEE*, vol. 21, no. 1, pp. 35–38, 2014.