Towards a File System Optimized for Scalable Media

Heiko Sparenberg, Volker Bruns, Seigfried Foessel
Moving Picture Technology
Fraunhofer Institute for Intregrated Circuits
Erlangen, Germany
{heiko.sparenberg,volker.bruns, seigfried.foessel}@iis.fraunhofer.de

ABSTRACT: *In professionel moviepost-production, various components are pushed to their performance limits. Former processors for example were not able to playback the JPEG 2000 compressed image-sequences which are used for distribution in Digital Cinema as well as long term storage in digital archiving – inreal time. While today's processors are able to decode such files a new bottleneck became part of the processing chain: In many cases conventional Hard Disk Drives (HDD) are not able to deliver the requested data – which is limited to a maximum value of 250 Mbit/s in Digital Cinema – in real time. In this paper we propose an algorithm for increasing the data throughput of conventional HDDs by utilizing the progression order of scalable media files, called UCODAS (Use-Case-Optimized DAta Storage). The motivation is given by the architecture of conventional hard drives, and finding that the file structure of scalablemedia, such as JPEG 2000, can be (re-)arrangedin such a way that the throughput of the disk can be significantly increased - especially if subsequent access patterns to the image-sequence are known a-priori. The advantages and disadvantages of today's hard drives are summarized before we show how scaling is achieved within JPEG 2000. Then, various methods for improving the performance of HDDs - taking advantage of the scalability - are proposed and the data sets used for the measurements are described. We performed tests using a collection of common file systems including FAT32, NTFS, ext2 and ext3 as well as RAW data access without a filesystem in order to prove our implementation of the UCODAS algorithm. In particular, we show that UCODAS can increase the data-throughput of a conventional HDD by more than a factor of 3 and thus overcome the bottleneck introduced by conventional HDDs.*

Keywords: Scalable Media, JPEG 200, Image Compression, Video Storage, Video Playback

Received: 17 September 2013, Revised 19 October 2013, Accepted 25 October 2013

1. Introduction

On one hand due to the architecture of storage devices, using rotating discs with a magnetic surface to store information permanently, random access is possible. On the other hand, hard drives require a certain time to navigate the read/write heads to a particular track storing the requested data [1]. This delay is referred to as Seek Time and since only in the most favorable case the requested sector is directly under the head when it starts to read a certain track, a rotational latency is incurred as well. The individual delays sum up and have a significant impact on the overall transfer rate of the HDD. In general, drives with rotating discs have a high throughput if data can be read continuously and search times for repositioning of the read/write heads are minimal. Disks, on which the requested records are fragmented and potentially spread over the whole device, show a

show a significantly worse performance.

The JPEG 2000 image compression standard [2] provides several scaling options, allowing the readout of different versions from a single code stream. Scaling of resolution, quality or color components as well as access to certain areas within a compressed image is provided. During compression, the data in a JPEG 2000 code stream is arranged according to a certain progression order. The progression order determines in what priority the individual packets of information are stored within the code stream. It is possible to produce images, where the first packets deliver all the information necessary to decode a low-resolution version of the image, but with highest quality (quality-oriented). By choosing a different progression order, data for the maximum resolution will be stored first, followed by subsequent packets that merely increase the quality (resolution-oriented). Packets can be rearranged even after the encoding process and without the need to decode the compressed image. Such a reordering causes a change of the progression while leaving the file size unaffected.

This property can be exploited to speed up hard drives, if the user behavior during subsequent file-requests of JPEG 2000 images is known a-priori. E.g. in the movie post-production, for many workstations used for cutting and conforming the resolution is limited by the attached displays. A resolution of 2K (2048×1080) suffices for a HD display (1920×1080), even if the maximum available resolution available from the stored JPEG 2000 files is 4K (4096×2160). Hence, it can be expected that most of the users only request the 2K portions of the images and skip the 4K parts.

Since HDDs perform best if the requested data can be read continuously, the JPEG 2000 images should be stored in a way, that the 2K portions can be read with minimal repositioning.

Therefore, the possibility to change the progression order shall be utilized when a sequence of images is written on a disk drive. Since we expect certain file systems to have an impact on the performance, a selection of common file systems was used for performance measurements in this work. In order to ease the simulation using different filesystems we implemented the UCODAS behavior as a standalone module, which is responsible for the re-ordering of the image files according to specific use-cases. Ideally, a file system driver would assume this task later on. Good candidates for such integration were presented in former works: A virtual file system for management of scalable media with the ability to define access rights to certain versions of a scalable media file was introduced in [8]. In [9], a real-time capable file system for scalable media files was presented. The focus of the latter work was the elimination of bottlenecks introduced by slow interfaces between storage devices and host systems.

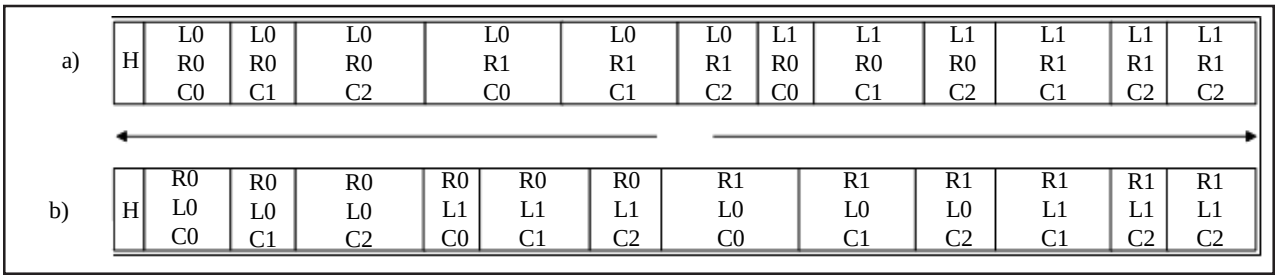

Figure 1. Structure of a JPEG 2000 image using a) LRCP-quality-oriented and b) RLCP-resolution-oriented progression

The utilization of other scalable media formats like H.264 Scalable Video Coding (SVC) [6] or MPEG4 Scalable to Lossless [7] is appropriate, but not in focus of this work.

The rest of the paper is organized as follows: Section II de- scribes the file structure and scalability features of JPEG2000. The description of UCODAS as well as its implementation is given in Section III and IV. Section V shows the gained results and Section VI presents the conclusion.

2. File Structure and Scalability Features of JPEG 2000

The JPEG 2000 compression mainly consists of a discrete wavelet transform (DWT) and the subsequent Embedded Block Coding with Optimized Truncation (EBCOT) entropy coder [3]-[5]. The application of the DWT results in individual sub-bands, which are divided into code-blocks of equal maximum size that are processed independently before they are stored in the final bit stream. According to the desired destination bitrate, code block information can be truncated as needed. This operation leads to a stronger quantization.

Scalability by resolution is a characteristic of the wavelet transform. Discarding code-blocks for the highest resolution-level and skipping the last synthesis step of the DWT reconstructs an image that is halved in both vertical and horizontal resolution. By skipping another synthesis step, the resolution is quartered once again and so on.

Scalability by quality is achieved by the introduction of so-called Quality Layers within the EBCOT stage. Here, each layer provides additional quality information for the already decoded sub-image. By discarding quality layers during the decoding process, the quality of the decompressed image is reduced. Spatial access to specific areas of an image is also possible since each code-block is coded independently.

In order to ensure a better navigation within the final code stream, several code-blocks are combined into precincts. Each precinct may be distributed across multiple packets in the resulting codestream. Finally, the complete code stream consists of multiple consecutive stored packets, including special marker segments that ease the navigation within the file or signal coding parameters.

Figure 1 shows two possible progression orders of a JPEG 2000 image comprising two quality layers (L0 and L1), two resolution levels (R0 and R1) and three color components (C0, C1 and C2): a) Layer-oriented progression order with priority to quality layers (LRCP). Here, all packets containing quality information will be stored first in the code stream. In order to read an image with reduced quality but in full resolution, only the first six packets have to be read from a storage device. The following six packets increase the quality for vertical and horizontal resolutions, b) resolution-oriented progression order with priority according to the resolution packets (RLCP). By reading the first six packets of the code stream, a lower resolution of the image with maximum quality can be decoded. Information for the next higher resolution levels is included in the remaining six packets.

3. Application-specific Ordering of Information Packets

The aim of the application-specific ordering is to ensure, that all JPEG 2000 files will be stored using a predefined progression order, regardless which progression order was used during compression. The selected progression order will be derived from access behavior that is expected when users request images from the HDD later. In this way, large data portions can be stored successively on the disk – a behavior that will improve the read-performance in terms of data-throughput since downtime for finding sectors is minimized.

The progression order of JPEG 2000 images is changed according to the selected behavior, when the data is copied to the storage medium. In this case, certain user behavior – defining for which use-case the HDD should be optimized – can be selected a) for the entire HDD during the format-procedure or b) on a per-folder basis within the management section of a special file system. Here, two strategies have been implemented, which are described in the following text.

3.1 File-oriented Reordering of Data Packets

To allow for sequential reads from disk later, data packets within a single file may be reorganized if necessary. This reordering leads to a change of the progression order on theone hand and allows for reading large chunks in a single uninterrupted turn on the other hand. Irrelevant packets will be skipped with one big leap that ends exactly at the position on the hard disk, where relevant packets of the subsequent imagebegin.In order to be able to restore the original files ifrequested,the native progression order has to be stored in the file systems' catalogue. Figure 2 shows this behavior. During the write operation, when the JPEG 2000 images are written to the

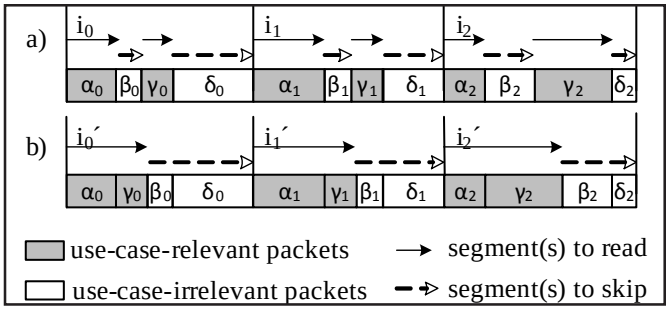

Figure 2. a) Original image sequence $i_0 - i_2$ to be stored on HDD with improper progression order for requested user behavior. b) Image sequence $i_0' - i_2'$ derived from original image sequence with new progression order

disk, the structure of the source images are analyzed (Figure 2a) and compared with the prescribed behavior. If the native progression order is not optimal for the predefined access strategy, it gets changed due to reordering of certain data packets (see Figure 1). Thus, all relevant packets for subsequent file-requests are stored sequentially on the disk before the non-relevant packets are stored (Figure 2b). Each arrow in Figure 2 represents a single independent request to the disk.

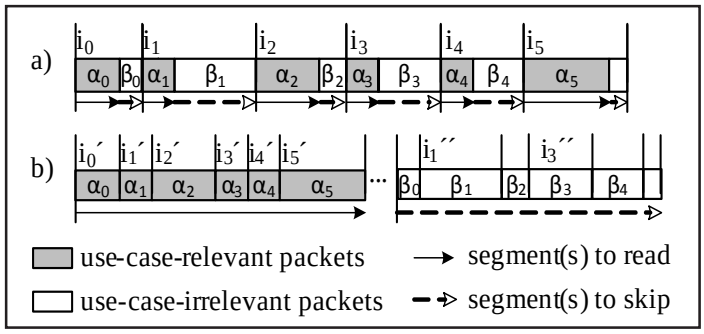

Figure 3. a) Original image sequence comprising relevant and irrelevant packets for certain user-behavior. b) Two sub sequences derived from original sequence according to sequence-oriented reordering of data packets

3.2 Sequence-oriented Reordering of Data Packets

By writing the relevant packets of all images sequentially to the hard disk, further optimization of the reading performance can be achieved. In this layout, each file will be divided into two or more parts, so that there is a relevant and a non relevant part. Again, non-relevant packets are those packets of the code stream that will most likely not be requested by a decoding application. The latter parts will be stored a) on a separate hard drive or b) on the same disk, after the relevant packets were stored sequentially. Figure 3a shows an image sequence with six images ($i_0 - i_5$), all comprising a relevant and a non-relevant part. Each image will be divided so that the relevant parts of all images can be stored sequentially on the hard disk. Figure 3b shows two different sequences: First, the relevant parts ($i_{0'} - i_{5'}$) and subsequently, all non-relevant parts of the original image sequence ($i_{0''} - i_{5''}$) for a particular access pattern.

A division of the original sequence into two sub-sequences - according to the progression order and the predefined user behavior - provides the advantage, that the entire sequence can be read without any repositioning of the read/write heads (compare arrow-quantity and lengths in Figure 2 and Figure 3). Again, the file system must save the metadata of the original file before the division so that a reconstruction to the original file-structure is possible later on.

4. Implementation

The performance of different file systems for different progression orders has been evaluated using the following data sets, which were generated using the current UCODAS implementation:

1. Image sequence with 120 images, 4K resolutions and layer-oriented progression order. The JPEG 2000 images were created using mathematically lossless compression; avg. file size is 10 MB.

2. 4K version of the same image sequence, where the progression order was subsequently changed from layer-oriented to resolution-oriented; avg. file size is identical to the layer-oriented image sequence.

3. 2K version of the same image sequence, which was derived from 2. by discarding all 4K packets (*R*1, see Figure 1); avg. file size is 6 MB.

All measurements were performed using a 250 GB HDD[1] comprising five empty partitions, each with a capacity of 10 GB. Four partitions were formatted with one of the following file system: FAT32, NTFS, ext2 and ext3. The fifth partition was used for direct read and write access to the hard disk without a file system (RAW). All records are accessible through a folder structure of the file system or by specifying a block index when using the RAW-mode.

Figure 4 shows the basic behavior of the software that was used for the performance measurements. Thealgorithmfirst reads the JPEG 2000 header of each file in order to determine the file structure, before it determines the relevant areas of the file to be read. Finally, all relevant parts of a file will be read from the selected partition. Figure 4 shows, that only the time for reading the requested packets was

measured – whereas the rest of the instructions for reading the JPEG 2000 structure etc., will not be considered when determining the read-performance.

The test-records were sequentially read from all partitions in the order described above. This operation was performed a hundred times before the ten best and worst measurements were discarded in order to minimize side-effects like onset of kernel processes (e.g. a start of a virus scanner or indexing service). Due to the total data size of all sequences and the fact that they were read from four partitions sequentially, before the first sequence was requested again, it is ensured that the operating system cannot cache the data.Rather, all data must be read from the disk. When accessing the data on the RAW partition using the current implementation,no caching of the operating system was involved.

It can be expected that the predetermined user behavior might change after the initial structuring was performed. For example, this would happen, if allworkstations wereequipped with a larger display while theachievable network throughput remains constant. In this case, a re-ordering of the scalable data according to the techniques described above may be useful. A lower quality version at full resolution could be read in a sequential manner instead. Subsequently, more quality information can be requested if required. Without further restructuring the data, the layout would not be optimal and the throughput of the disk would significantly decrease due to new user behavior.

```
01: Algorithmread_sequence (path, packetsToSkip)
02:  for all images img_i do
03:    j2kImg = parseJPEG2000 (img_i );
04:    relevantParts = getRelevantParts ( j2kImg, packetsToSkip);
05:    for all parts p_i do
06:    start = getSystemTime ();
07:    readPart ( p_i );
08:    stop = getSystemTime ();
09:    bytesRead += p_i.totalSize;
10:    processingTime += (stop – start);
11:  endfor
12: endfor
13: return processingTime;
```

Figure 4. Pseudo code for time measurement function taking source folder path to series of JPEG 2000 files

Fortunately, thanks to the properties of scalable media, a subsequent reordering is possible without the need to re-encode the images. For this, the data must be restructured again so that it is optimally laid out for the new user behavior. To evaluate the time and system performance required for a realignment of the files we further examined the file-oriented approach (3.A). The algorithm used for our measurements is shown in Figure 5. In principle, image data is read from disk, restructured in memory depending on the new progression order (`newProgOrder`) and saved back to the same sectors on the disk. Again, each step is measured by using the system clock in order to subsequently determine the performance.

```
01: Algorithmrestructure_img ( path, newProgOrder)
02: readStart = getSystemTime ();
03: j2kImg =ReadFile (rawDevice, startIndex, length);
04: readStop = getSystemTime();
05: processing_start = getSystemTime();
06: j2kImg = change_prog_order ( j2kImg, newProgOrder);
07: processing_stop = getSystemTime ();
08: writeStart = getSystemTime ();
09: WriteFile (rawDevice, j2kImg, startIndex, length);
10: writeStop = getSystemTime();
11: return (readStop – readStart) + (processing_stop – processing_start) + (writeStop – writeStart);
```

Figure 5. Pseudo code for restructuring algorithm including time measurements

Even after arestructuring, theseqence must bestoredon the disk in a sequential fashion so that successiveimages can bereadin a single turn without expensive repositioning of the read/write heads. To ensurethis,we have chosenthe RAWaccess method for our measurements, since it is the only method that allows for the restructered data to be writtentothe exactsame sectorson the storage device. Thus, we can ensure that images remain sequentially stored on the one hand.On the other hand, using one of the file systems listed above there isno way to defineat which exact position on the disk the data gets stored.Since the file system is free to store data anywhere on the disk, the images couldbescatteredall over the driveafter reconstruction, even ifthey werepreviouslystoredoptimallyfor fluent playback.

5. Results

According to the use case described in section 1, requesting processes are only interested in the 2K version of the files – no matter in which way the files were stored on the HDD. Figure 6 shows, that application of UCODAS can achieve significant performance improvements when data is requested from a hard disk drive. This simulation reads a 2K resolution from an image sequence with a resolution of 4K. First, the quality-oriented data is read without reorganization. According to Figure 2 several repositioning steps are necessary in order to read the 2K versions of the files. All tested file systems achieve a comparable performance of around 24 MB/s. This value is well below the maximum data rate of the hard drive, which was measured using drive performance software [10] to be around 92 MB/s.

By resorting to a resolution-oriented progression order within each file, the data throughput increases significantly since the required time for repositioning the read/write heads is reduced. The performance increase varies depending on the tested file systems. A particularly high increase can be observed for FAT32 and NTFS (about 73 MB/s). The ext2 and ext3 file systems also show a higher data throughput, but not as high as FAT32 or NTFS (about 61 MB/s).

A similar effect can be observed when reading the third test set: The largest increase in performance can be achieved with the FAT32 file system which delivers the data at about 82 MB/s. NTFS and ext2 provide a minimal increase in performance to a total of 63 MB/s. When accessing the hard disk directly, without employing a file system, distinct transfer rates can be achieved.Especially the access to the third image sequence shows a further increase in performance compared to an access using a file system. The measured average throughput of 90 MB/s almost corresponds to the maximum measured hard drive performance. The remarkable result of RAW-access can be explained by the uncompromising sequential organization of the image files. This property is not necessarily given when using one of the tested file systems.

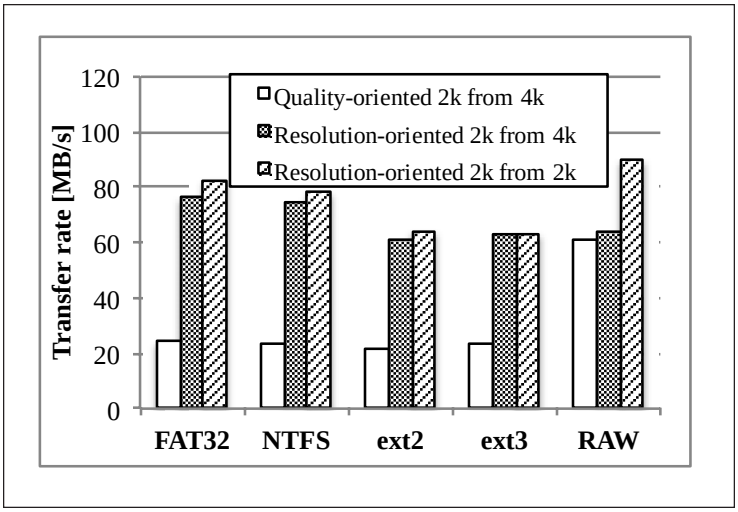

Figure 6. Data throughput measured using different file systems as a function of file-inherent progression order

In addition, we measured the time-expenses for changing the internal structure of the scalable files in response to a change of the user-behavior. The measurements were performed using the test set described above. The progression order was changed from LRCP to RLCPandCPRL. The measured results for the new progression orders were almost identical, so that only the measured percentages for the restructuring from LRCP to CPRL are shown inTable 1. It is found that only 4% of the overall process is required for the restructuring of the image data in memory. 96% of the time is spent for reading and writing the data. The results show that a

subsequent change in the structure can essentially be regarded as a single read andwrite operation perfile. The data throughput of the HDD during the restructuring process was measured to be approx. 92MB/s.

Process	Percentage time consumption
Read from HDD	48%
Restructure Image	4%
Write to HDD	48%

Table 1. Comparison of the required processing times for the main steps of the reorganization of existing data

6. Conclusion

In this work we presented UCODAS, a use-case optimized storage technique for scalable media, an implementation of this algorithm as a standalone module, performance measurement as a function of four common file systems as well as the implementation of RAW access to the storage device in order to measure the performance decrease in data throughput introduced by a file system. Applying a use-case-optimized progression order provides a useful way to increase the throughput of hard disk drives significantly. Especially the specific adaptation of the progression order within a file shows a) a significant increase in data throughput and can b) be changed quickly by another reorganization of the scalable JPEG 2000 image sequence, e.g. if the predefined use caseneeds to be adjusted due to changed user behavior.

Since the file-size of the images is not affected when changing the progression order, code streams can easily be rearranged on the already occupied space on the hard disk.

Splitting images and storing parts of JPEG 2000 files to different areas of a hard drive can further improve the overall performance on the one hand. On the other hand, the original version of a sequence can only be read with a low performance since the disk's read-heads have to jump across the platter several times for each image, depending on the file structure and the length of the sequence. Storing the irrelevant data packets on a separate hard disk can prevent this. However, re-assembling the original sequence can be realized very efficiently using a special copy-process that allocates memory for the complete file on the target disk, copies all relevant parts of the reorganized version before it copies the irrelevant parts. Development of these copy-algorithms as well as an investigation of the performance of the presented algorithms for other scalable media formats like H.264 SVC or MPEG4 SLS will be subject to further research.

References

[1] Karl Paulsen. (2011). Moving Media Storage Technology, Elsevier, Burlington.

[2] ISO/IEC 15444-1. Information technology – JPEG 2000 image coding system – Part 1: Core coding system.

[3] Taubman, D. S., Marcellin, M. W. (2002). JPEG2000, Image Compression Fundamentals, Standards and Practice, Kluwer Academic Publishers, Boston.

[4] Acharya, T., Tsai, P. (2005). JPEG2000: Standard for Image Compression, Wiley, Chichester.

[5] Taubman, D., Rosenbaum, R. (2003). Rate-distortion optimized interactive browsing of JPEG2000 images, *In*: Proc. IEEE International Conference on Image Processing, 3, p. 765–768, Sep.

[6] Schwarz, H., Marpe, D., Wiegand, T. (2007). Overview of the Scalable Video Coding Extension of the H.264/AVC Standard, *IEEE Transactions on Circuits and Systems for Video Technology*, 7 (9), 9 Sep.

[7] Yu, R., Geiger, R., Rahardja, S., Herre, J., Lin, X., Huang, H. (2004). MPEG-4 Scalable to Lossless Audio Coding, Audio Engineering Society, San Francisco, 28 October.

[8] Sparenberg, H., Schmitt, A., Scheler, R., Foessel, S., Brandenburg, K. (2011). Virtual File System for Scalable Media Formats, 14th ITG Conference - Dortmunder Fernsehseminar.

[9] Sparenberg, H., Martin, M., Foessel, S. (2012). Real-time Capable File System for Scalable Media, *Picture Coding Symbosium*.

[10] EFD Software. (2012). HD Tune, http://www.hdtune.com/, December 26.