Algorithmically optimized AVC video encoder with parallel processing of data

Tomasz Grajek, Damian Karwowski, and Jakub Stankowski

Poznan University of Technology,
Chair of Multimedia Telecommunications and Microelectronics
{tgrajek,dkarwow,jstankowski}@multimedia.edu.pl

Summary. Algorithmically optimized AVC (MPEG-4 part 10 / H.264) video encoder with parallel processing of data is presented in the paper. The paper reveals the architecture of the proposed encoder together with the description of the applied software optimization techniques. Conducted experiments show exactly the degree of parallelization of computations in the encoder and the compression performance of video encoding.

Key words: optimized encoder, AVC video compression, MPEG-4 part 10, H.264, encoding efficiency

1 Introduction

Hybrid video encoders are of a great importance in communication systems due to their ability to represent a video on relatively small number of bits. The most popular hybrid encoder that is currently used in areas such as IPTV and high definition (HD) television is AVC (MPEG-4 part 10 / H.264) video compression [1]. This technique is well known and has already been described in many papers and books [2, 3]. As a matter of fact the newer technique of video compression has been worked out recently (called High Efficiency Video Coding – HEVC) [4], but the application of the new technique is an issue of further future. Therefore, the AVC technique is also the subject of ongoing studies.

The AVC allows 100 fold reduction of the encoded data stream for high quality of the encoded video. Such a strong compression of a video is possible by the use of the advanced intra- and inter-frame predictive coding of image blocks together with the sophisticated methods of transform and entropy coding of residual data [2, 3]. Nevertheless, applied in the AVC algorithms make both the encoder and the decoder computationally very complex. In the case of the HD videos and AVC, the real-time video encoding is a big challenge even for todays' high performance multimedia processors. It is particularly

true for the video encoder side, whose complexity may be dozens of times greater than the complexity of the decoder. From that reasons, an important research problem is finding of such an architecture of highly optimized AVC compliant video encoder that will be able to exploit the potentials of todays' multimedia processors in an efficient manner. This makes the topic of this paper.

2 Research problem

The architectures of optimized AVC encoders have already been the topic of the authors' previous research. As a result of this works the optimized structure of AVC video encoder was proposed, that was dedicated to x86-based platforms [5]. High computational performance of the encoder was achieved performing algorithmic optimization of encoder functional blocks, taking into account both the specificity of the AVC algorithms (application the context-based coding mechanisms) and features of x86-platforms (ability of using the vector operations and small size of fast cache memory). Although the proposed architecture increased the throughput of video encoder 77 times on average [5] (relative to the JM 13.2 reference software of AVC and at the virtually the same compression performance of the encoders) the encoder proposed earlier was intended for a single processor platforms with sequential processing of video data in general. This paper makes the continuation of the previous works and concerns the tuned version of the optimized AVC software encoder with additional possibility of performing computations in parallel by the use of multi-core/multi-threading technologies. It must be emphasized that parallelization of computations in the AVC encoder is a very difficult technical problem, especially in the context of the software realization of the encoder. It is commonly known that the using of the context-based coding paradigm in AVC (the use of the previously encoded data for encoding of the data of the currently processed image block) together with the raster scanning order of image blocks force in a large extent sequential processing of image data. The goal of the paper is to explore the possibilities of concurrent application of multiple processors in the encoder, analysis of such an encoder throughput and compression performance of video encoding. It must be stressed, that the assumed constraint is to preserve the full compatibility of the developed encoder with the AVC standard. The obtained results will be referenced to those obtained for the sequential version of the optimized AVC encoder described in [5].

3 Architecture of optimized AVC encoder with parallel processing of data

3.1 Introduction

As stated before, the architecture of the optimized, sequential version of the AVC encoder (that was proposed by the authors' earlier [5] was the starting point for works on the version of the encoder that is capable to perform computations in parallel. At the course of the study the authors found out that a possibility of dividing the frame into independent fragments called slices is the only real way for doing computations in parallel in the software version of the encoder. Since each slice is a self-contained unit which content can be fully encoded and decoded without referencing to data of other slices [2, 3], the individual slices of the image may be processed in parallel with the use of multiple processors (or processor threads) at the same time.

Therefore, division of the images into multiple slices is the main solution applied in the proposed video encoder. In this solution, the degree of computations parallelization in en-coder depends directly from the number of slices within an image and the number of processor cores (or threads) available in the system. Nevertheless, not all of the functional blocks of AVC encoder can be parallelized in this way. Notable exception is deblocking filter and image interpolation procedure which, from the reason of data dependencies, operates on the entire image.

3.2 Parallel processing of slices

In order to get the possibility of parallel processing of data in encoder, two parts were extracted in the structure of the encoder. These are: 1) the management part, and 2) the execution part. It was realized in such a way that two groups of encoder program threads are created: master thread and a slave thread(s). There is only one master thread, while there may be a higher number of slave threads. The master thread is responsible for controlling the work of video encoder, that is to say allocation of memory buffers, assignment of individual structures to objects, controlling the global bitrate and the bitrate for individual images, dividing the image into slices, and finally running the slave threads. Whereas, the individual slave thread deals with encoding of assigned image slice, or the entire image (depending on configuration of the video encoder). In case of one slave thread and a single slice within an image (see Fig. 1), the master thread prepares all the data for encoding and runs the slave thread. After that, the master thread waits until the slave thread will encode the image and finish the operation. Then, the master thread performs deblocking filtering of decoded image, and does the image interpolation to 1/2 sampling period (only for the reference images), and manages the encoded image (i.e. inserting the image into the reference image list, writing the bitstream to the output data stream, etc.).

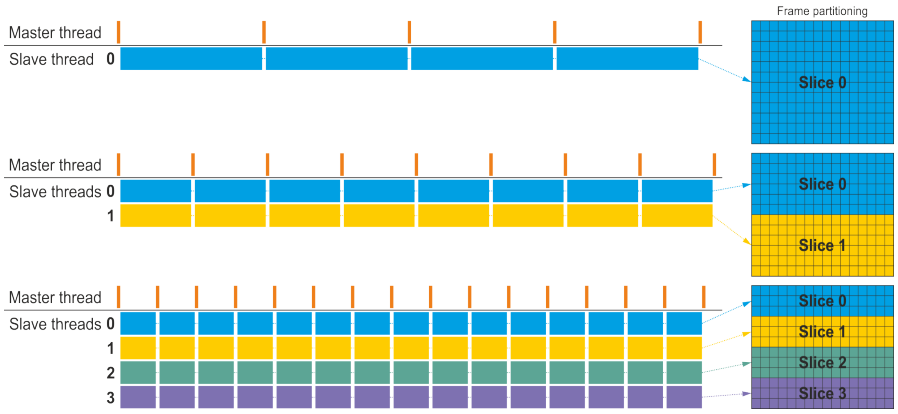

Fig. 1. Parallel processing of data in optimized AVC encoder.

In case of two slave threads and two slices within a frame, the master thread prepares the relevant data for the first and the second slave threads and runs them. Each of the two slave threads encodes the image slices that were assigned to them. It must be emphasized that the slave threads work in parallel, due to the fact that they operate on a separate data sets. The master thread waits until the slave threads will finish the encoding and proceed with deblocking filtering and image interpolation (if necessary).

In the scenario of fewer number of image slices in comparison to the number of slave threads, some of the slave threads will not be utilized. In the opposite situation (the number of image slices exceeds the number of slave threads) when a given slave thread will finish the encoding, it will get another task of encoding the next image slice from the master thread. This process will continue until all image slices will be encoded.

In the situation when at least two image slices are concurrently encoded, typically there is a problem of simultaneous access of individual slave threads to a common memory. It is especially critical when at least one of threads tries write data to memory area that is used by another thread. In order to avoid such memory access conflicts, encoder uses separate memory cache for each thread and avoid race conditions by allowing slave threads only to read from common memory.

4 Methodology of experiments

Coding efficiency of the optimized AVC encoder with parallel processing of data was thoroughly investigated with set of test video sequences. The goal was to explore the influence of allowed number of slices and allowed number of processor threads on encoding speed and efficiency. Experiments were done according to the following encoding scenario:

- Full HD test video sequences were used: *BasketballDrive*, *BQTerrace*, *Cactus*, *Kimono1*, *ParkScene*. The sequences were recommended by groups of experts ISO/IEC MPEG and ITU VCEG as a test material for research on new video compression technologies [6].
- Structure of group of pictures (GOP) was set to IBBPBBPBBPBBPBBP.
- Experiments were done for a wide range of bitrates (controlled by QP = 22, 27, 32, 37). This results in the quality of a reconstructed video from excellent (QP=22) to very poor (QP = 37).
- Allowed number of slices and allowed number of processor threads was set to 1, 2, 4, or 8.
- CABAC entropy encoder was used.
- Testing platform: Intel(R) Xeon(TM) CPU 3.06 GHz (6 cores, 12 processor threads, Nehalem microarchitecture), 24 GB RAM, Windows 7 64-bit.

During experiments all combinations of number of slices and number of processor threads was tested.

Dividing frame into slices results in slightly different bitstreams (from the viewpoint of their size and quality of reconstructed videos). In order to compare results achieved for different number of slices and number of processor threads settings the Bjøntegaard metric was calculated [7]. The metric allows to compare the RD curves of two encoders in terms of bitrate reduction and PSNR gain based on four RD points (for QP = 22, 27, 32, 37 in experiments). Such tests were done for luma (Y) component. It should be noted, that the setting one slice per frame and one processor thread results in pure sequential encoder. The pure sequential encoder provides a benchmark for the performance of the parallel encoder.

5 Results

First of all we have evaluated influence of dividing frames into slices on encoding efficiency. Achieved results have been gathered in Table 1. Dividing frame into slices results in increase of the bitrate on average on 0.60%, 1.68% and 3.50% for 2, 4 and 8 slices respectively in comparison to case with only one slice per frame. Obtained results are consistent with those reported in [8].

Figure 2 presents achieved encoding speedup for all combinations of investigated number of slices and processor threads for exemplary video sequence. Achieved speedup depend on target bitrate (controlled by QP) due to different ratio between parallezized slave thread(s) tasks and sequential master thread processing.

Table 2 present results supplemented with achieved speedup resulted from using more processor threads (i.e. how many times encoder using N processor threads is faster that encoder with only one processor thread allowed). The first observation is that number of allowed processor threads dose not influence total encoding time in case of one slice per frame. The negligibly small increase

Table 1. The average bitrate change (Bjøntegaard metric) resulted from dividing frame into 2, 4 and 8 slices per frame against 1 slice per frame. Positive numbers correspond to bitrate increases.

Sequence	2 slices per frame	4 slices per frame	8 slices per frame
BasketballDrive	0.98%	2.21%	4.76%
BQTerrace	0.69%	1.81%	3.30%
Cactus	0.31%	1.01%	2.18%
Kimono1	0.86%	2.47%	5.25%
ParkScene	0.22%	0.89%	2.03%
Average	**0.60%**	**1.68%**	**3.50%**

of total encoding time for four and eight processor threads results from thread management overhead. For two slices and two and more processor threads we achieve 1.64 times faster encoder that for one slice case. For four slices and four and more threads the achieved speedup is about 2.5 and for the last case (eight slices and eight processor threads) - 3.02.

Achieved results clearly confirm that some part of the encoder has not been parallelized. Otherwise, dividing frame into two slices and using two processor threads would result in approximately two times faster encoder.

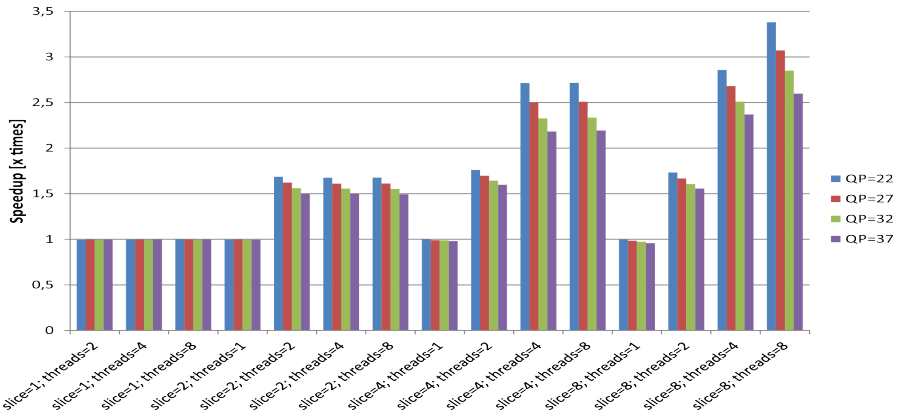

Fig. 2. Speedup (execution time of analyzed case to one slice with one thread case ratio) for *BasketballDrive* sequence for different QP values.

6 Conclusions and final remarks

The presented results prove that the use of the idea of frame slices can be the basis for substantial parallelism of computations in the video encoder. The exact

Table 2. The average bitrate change (Bjøntegaard metric) and average speedup according to one thread with one slice per frame case. Positive numbers of DB rate correspond to bitrate increases. Speedup means how many times the execution time of analyzed case is lower than for one thread with one slice case.

DB-rate / speedup	1 slice per frame	2 slices per frame	4 slices per frame	8 slices per frame
1 thread	**0.00% / 1.00x**	0.60% / 1.00x	1.68% / 1.00x	3.50% / 1.00x
2 threads	0.00% / 1.00x	**0.60% / 1.64x**	1.68% / 1.63x	3.50% / 1.63x
4 threads	0.00% / 0.99x	0.60% / 1.64x	**1.68% / 2.51x**	3.50% / 2.51x
8 threads	0.00% / 0.98x	0.60% / 1.66x	1.68% / 2.54x	**3.50% / 3.02x**

impact of the number of processor threads and the number of slices in the image on the degree of computations parallelisation in the encoder was numerically presented in experimental section. In an exemplary scenario of four slices and four processor threads the encoder works 2.5-times faster relative to the sequential encoder. Taking into consideration the algorithmic optimizations that had been previously carried out in the sequential version of the encoder (see [5] for more details) gives almost 200 times faster encoding with respect to the complexity of the JM 13.2 reference software.

Since the use of N image slices (with the same number of processor threads) does not give in general the N-times faster encoder (see experimental results) there exists parts of the encoder that were not parallelized. Detailed analysis of the operations that are carried out in the encoder allowed for identification of such a fragments. These are mainly: image interpolation and deblocking of reconstructed images. Further optimization of these functional blocks can be a source of additional acceleration of the encoder. The results revealed also that the compression performance loss (expressed as BD-Rate increase) will not exceed 3.5% in the scenario of 3-times encoder acceleration.

Acknowledgment

The presented work has been funded by the Polish Ministry of Science and Higher Education for the status activity consisting of research and development and associated tasks supporting development of young scientists and doctoral students.

References

1. ISO/IEC 14496-10: Generic Coding of Audio-Visual Objects, Part 10: Advanced Video Coding. March 2010.
2. I.E. Richardson, The H.264 Advanced Video Compression Standard. Second Edition, John Wiley and Sons, 2010.
3. Special issue on H.264/AVC video coding standard. IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, July 2003.

4. ISO/IEC 23008-2 (MPEG-H Part 2) / ITU-T Rec. H.265: High Efficiency Video Coding (HEVC), Apr. 2013.
5. T. Grajek, D. Karwowski, A. Łuczak, S. Maćkowiak, M. Domański, Architecture of algorithmically optimized MPEG-4 AVC/H.264 video encoder, Lecture Notes in Computer Science, no. 7594, pp. 79–86, Springer-Verlag (Proceedings ICCVG 2012 Warsaw, Poland, September 24-26, 2012).
6. F. Bossen, Common test conditions and software reference configurations ", Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11, Doc. JCTVC-J1100, Stockholm, Sweeden, Jul. 2012.
7. G. Bjøntegaard: Calculation of Average PSNR Differences between RD curves. ITU-T SG16/Q6, 13th VCEG Meeting, Austin, USA, April 2001, Doc. VCEG-M33.
8. V. Sze, A.P. Chandrakasan, A High Throughput CABAC Algorithm Using Syntax Element Partitioning, IEEE International Conference on Image Processing (ICIP) 2009.