

# An Exploration of the Hardware Implementation of Quadtree Compression

Michiel D'Haene, Hendrik Eeckhaut, Mark Christiaens, Dirk Stroobandt  
Ghent University, ELIS  
Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium  
{mdhaene, heeckhau, mchristi, dstr}@elis.ugent.be

*Abstract*—The 2D-wavelet transform of images is, among others, used for image compression (either for still images or for video sequences). A wavelet transformed image has the special property that the image content is split into a low-pass component and a high-pass component. The high-pass component usually consists mostly of small coefficients and can be compressed efficiently using an entropy coder. A Quadtree coder [9], [6], [5] is such an entropy coder which achieves good compression rates and in addition has a number of scalability features. In this article we present the results of a hardware implementation of the Quadtree algorithm. We find that the progressive coding and the reordering of the video stream during Quadtree coding requires a tailored approach to its hardware implementation in order to achieve an efficient memory use and a high compression speed. We present a number of techniques to deal with these challenges and give simulation results of the expected performance of the Quadtree coder. Compared to the original algorithm in software, processed on a PentiumIV 2,4 Ghz, we reach a speedup by a factor of 9.

*Keywords*—wavelet transform, Quadtree, hardware implementation

## I. INTRODUCTION

An image is said to be coded in a scalable fashion when it is possible to obtain a version of this image with reduced resolution or reduced quality (see Figure 1) without actually having to decode the entire image. This property is extremely important in a resource constrained environment such as PDAs, cell phones, ... since scalability allows these devices to perform less computations and still achieve a reasonable quality level.

If we look at conventional DCT-based<sup>1</sup> algorithms then we see that these can achieve good compression results but have serious difficulties when coping with these scalability requirements. Not only do they require the entire image to be decoded but moreover the quality of these images becomes very poor when used at low bitrate-levels. This is due to the block-based nature of these encoding schemes resulting in clearly visible block artifacts [10].

Scalable, wavelet-based video coders try to address this

<sup>1</sup>Discrete Cosine Transform



Fig. 1. Scalability in quality (above) and in resolution (below) applied to the Lena image.

problem. A 2D-wavelet-transform has the special property that the image content is split into a low-pass component and a high-pass component. The high-pass component usually consists mostly of small coefficients and can be compressed efficiently using an entropy coder. Wavelet coding provides us with out-of-the-box resolution scalability: simply drop some of the high-pass layers of the wavelet image and decode the remainder of the image. The resulting decoded image is a scaled-down version of the original. Quality scalability can be obtained through the entropy-coder. If you structure the encoded wavelet image in such a way that additional bits of the wavelet coefficients are stored in different parts then by removing some of these parts you obtain an image with reduced quality.

There is nevertheless a downside to this type of codec: due to its increased complexity it requires a much higher level of computational power. The wavelet transform itself can be performed efficiently but the entropy coding is a very expensive operation. To achieve real-time decoding we must turn to hardware acceleration. In this paper we present our experience and results regarding a fast and efficient hardware implementation of an existing entropy coder on an Altera PCI Development Kit, Stratix Edition. This board contains among others an Altera Stratix, EP1S25 FPGA and 256 MiB memory.

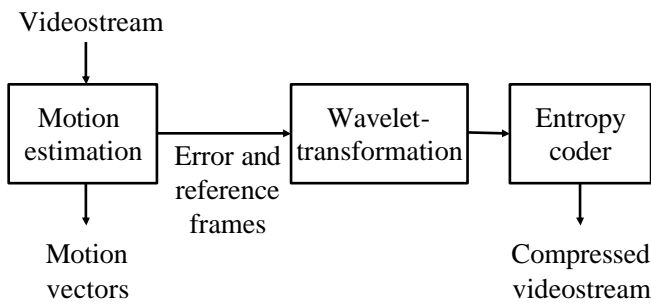


Fig. 2. The wavelet-based video coder

## II. OVERVIEW: THE ENTROPY CODER

In this paper we implement the “Quadtree” entropy coder, developed as a part of the wavelet based video coder described in [5], in an FPGA. In Figure 2 a global overview of this coder is presented. The wavelet coder consists of three components.

The first component performs motion estimation of images [1]. The results of this component are motion vectors and error and reference frames. Motion vectors try to approximate an image as accurately as possible using parts of other images as a reference. The error frame is the difference between the original frame and the reconstructed frame.

The second component performs a discrete wavelet transform of the images [3].

Finally the entropy coder compresses the wavelet transformed images by exploiting statistical properties of the wavelet image. The entropy coding is accomplished in three stages: a “Quadtree” division, a context based model selection and an arithmetic coder.

The Quadtree division forms the principal component of the entropy coder. This component tries to reduce the spacial redundancies in the images. To do this, it performs a recursive division of each image, until the subdivisions have a more or less homogeneous structure. The recursive division performed is very simple: if a certain condition is fulfilled, the image is divided into four sub quadrants. Then the condition is tested again in each quadrant, etc.

At the end of the division process, the created divisions can be coded efficiently in the next two steps. Moreover, the division process is being applied in a progressive manner to the image i.e. first the most significant digits of the wavelet coefficients are being encoded, followed by the less significant digits. This gives the decoder the opportunity to scale the image in quality without the necessity of decoding the whole bit stream. As soon as the decoder has sufficient accuracy, the decoding process can be aborted. Because the information with high contribution to

the image quality is shifted towards the start of the video stream, the quality of the image, given the decoded information, will be optimized even for low bitrates. Also due to the absence of block based compression there are no block artifacts in low quality images.

The next stage is the context based model selection. This stage is located between the Quadtree division and the arithmetic encoder. The context based model selection continuously collects statistical information about the image. This information is then used to forward newly received information into different bit streams. We call these bit streams “context models”. The idea is that the stream of bits of one context model consists of bits with the same statistical properties (i.e. the same probability of the occurrence of a bit with value 1). For example assume that the eight neighbors of a certain coefficient have a very small value. Taking into account the smoothness of images we know that the probability of the coefficient to also be small is very high. To allow the context based model selection to make such predictions a history is maintained.

Thanks to the preliminary sorting of information in stage 2, it is possible in stage 3 to compress the information very efficiently using a simple adaptive arithmetic encoder [4]. The compression achieved surpasses that of dictionary based algorithms. An arithmetic encoder internally keeps a statistical frequency model of the symbols to perform the compression. The adaptive model changes based on the contents of the data input stream and this way automatically converges to an optimal compression rate [8].

The arithmetic encoder used in this video encoder is a simple  $0^{th}$ -order implementation [7] with one bit as symbol unit i.e. it does not take into account the inter symbol dependencies. This drastically reduces the memory usage. Instead of a higher order implementation, each context model uses his own independent frequency model. This way, each context model can be compressed with an individual optimized statistical model. The combination of the context based model selection with the arithmetic encoder gives rise to very high compression rates. An overview is presented in Figure 3.

The most important obstacles encountered during the implementation of the entropy encoder were caused by the highly sequential nature of the algorithm (this phenomenon is typical for compression algorithms), the extensive memory requirements and the difficult cooperation between the different blocks. In the next session we give an overview of the changes made to the algorithm, necessary to come to an efficient hardware implementation.

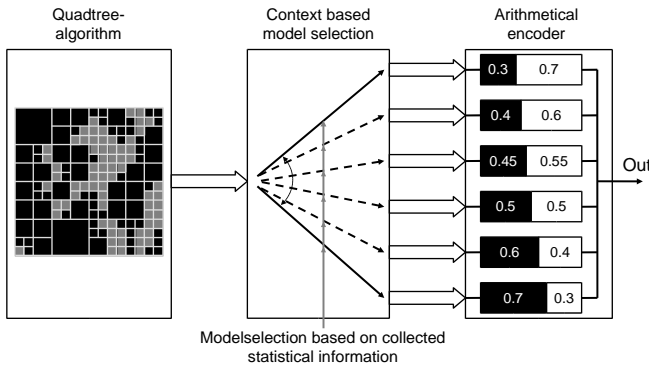


Fig. 3. Overview of the entropy coder

### III. CHANGES MADE TO THE CODER

#### A. The Quadtree Division

The condition used by the Quadtree to decide whether or not a quadrant should be further subdivided is based on a layered approach. First the top layer of the wavelet image is located by looking for the coefficient with the largest absolute value. This value becomes the current threshold. During successive iterations this threshold is lowered and all the coefficients which have a larger value are encoded, the others are ignored. Eventually, the threshold is lowered to 0 and the entire wavelet image is encoded. During this traversal the Quadtree uses two lists: “the refinement list”, storing the positions of coefficients that became significant (i.e. have a value higher than the current threshold) and the “non significant list” storing the positions of coefficients having a high probability to become significant during the next iteration [6]. In the original code, these lists were built up as a 2D-structure representing the considered image. This is shown in Figure 4a.

Due to the size of this 2D structure it is impossible to implement this data structure directly in the fast on-board SRAM of the FPGA. Furthermore due to the highly random access pattern to this data structure it is unwise to store it off-chip in external memory. So we need to modify this data structure to achieve good performance.

A first important improvement to the code was to rewrite the memory structure from a pointer-oriented 2D-structure to an array-based 1D-structure. Because the lists are always traversed from beginning to end and new elements are only added at the end of the list, the access to the memory can be made much more predictable by placing elements sequentially in the memory. This way the use of pointers becomes unnecessary, also resulting in a significant decrease of the required memory to store this structure. Moreover, the memory has been hierarchically organized: the complete list is stored in the external DDR-

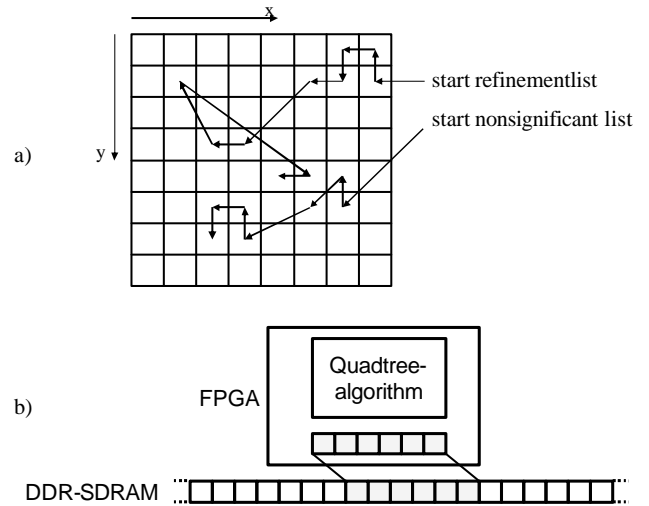


Fig. 4. a) the 2D linked list structure of the original coder, b) the 1D structural list of the reworked coder

memory, while a little fraction of the list is cached in fast, internal SRAM on the FPGA (Figure 4b). By implementing a smart preload function, calculations can be performed at full speed, without waiting for data from the external DDR memory.

A second problem we addressed is the recursive nature of the subdivision process of the Quadtree. We implemented an optimized stack module that takes care of storing the necessary information for a new iteration. Also it stores some extra information as will be explained next.

An important improvement was made to the iterations themselves. After each recursive subdivision, a part of the calculations had to be repeated inside each quadrant. By deriving some extra quadrant specific information during each calculation, the needed computational power can be decreased drastically ( $< 25\%$  of the original needs). The information is then stored together with the division information on the stack.

Finally, the image itself has to be accessed as fast as possible. Thanks to the layered approach of the Quadtree division, it is sufficient to store just one layer in fast internal memory. Again, a preload function takes care of new image information to be present in the internal memory in time.

This way, we obtain a processing speed of 99 QCIF<sup>2</sup> images and 24 CIF images per second using an FPGA clock speed of a 100 MHz. Compared to the original algorithm in software, processed on a PentiumIV 2.4 Ghz, this means a 9 times speedup.

<sup>2</sup>Quarter Common Intermediate Format,  $176 \times 144$  pixels

## B. Context Based Model Selection

The context based model selection has the principal job to sort information into different context models, using image information collected during processing. Due to the high amount of information to be stored, this block consumes by far the largest portion of memory so the use of DDR memory is inevitable.

The internal structure of storage has been adapted to allow a more compact representation of the necessary information and to allow a more efficient communication with the external DDR memory (use of DMA to transfer large chunks of data at a time). This way we can assure that this block is fast enough for the Quadtree subdivision process. However, the basic idea of the algorithm can be improved drastically from the viewpoint of a hardware implementation. In the original code, two models are kept for the information: one model with old information used for the prediction of new data, and one model containing the updated information. This involves also a copy action after each cycle. By combining the two models into one continuously adapted model, the memory requirements can be reduced by 50%. This would require a modification of both the encoder and decoder and has not yet been implemented.

## C. The Arithmetic Coder

Finally the arithmetic encoder takes care of the real compression of the bit stream. This stage turns out to be the slowest stage of the encoder. There are many dependencies in the code, thwarting a parallelized approach: each symbol has to be coded using results from earlier coded symbols. This causes an inherent sequential coding scheme.

To obtain maximal performance, the code has been reworked into a complex pipeline, as shown in Figure 5. This allows us to compress a symbol in approximately 6,35 clock cycles at 100 MHz. Converted to frame rates, this means 18,9 CIF images per second.

Other possibilities for further speedup are for example cumulating the model adaptation operations over two or more symbols reducing the calculation cost. Another possibility is the use of some static models (i.e. which predict a fixed probability) instead of adapted models. Some models will have a constant statistical behavior in time. So it will not have negative influence on the final compression quality if we use a fixed model for them. Taking these improvements into account, real-time decoding of CIF images (30 frames per second) will become possible.

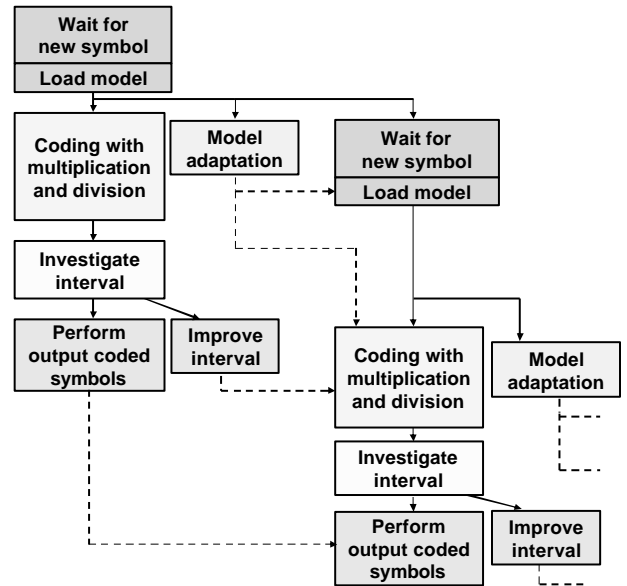


Fig. 5. Simplified schematic of the pipelined version of the arithmetic encoder

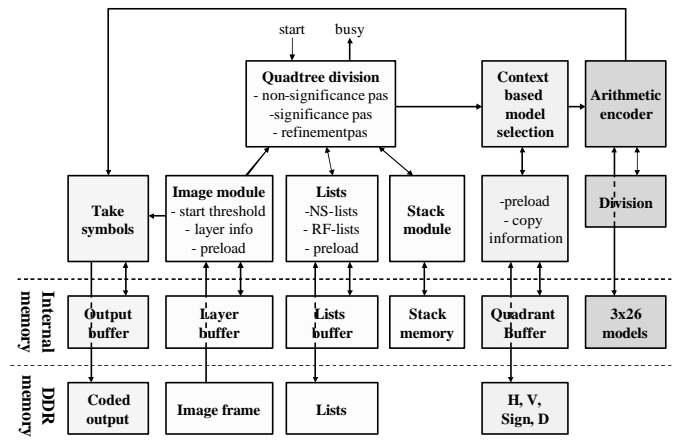


Fig. 6. Global overview of the reworked entropy coder

## D. Putting It All Together

In Figure 6 a global overview of the reworked entropy coder is presented. Due to the different processing speed of the first two blocks, a simple coupling of the three modules without any buffering will result in a big performance decrease ( $> 50\%$ ). While the Quadtree subdivision is calculating it does not produce data, so the model selection has nothing to do. Once the Quadtree is finished with the division process, it produces a big amount of data at once. At this time, the context based model selection has to begin calculating which models to use before it can effectively start processing the data. Meanwhile, the Quadtree has to wait until the model selection has processed the symbols.

An adequate buffering of these two blocks would re-

quire an unfeasible amount of memory. Another approach is to implement an opcode-operand structure in the symbol queue. Although it reduced the memory amount, it increases the complexity of the coder. Some other possibilities to improve the performance of the coder need functional changes in the algorithm. For example, there is some inefficiency in the manner the Quadtree is constructed. An improvement of the algorithm will also reduce the computational efforts and from this smooth out the processing of data in the first two blocks.

#### *E. Conclusions and Future Work*

Despite all these improvements, a coder for coding CIF images in real time remains difficult and needs some inelegant tricks. For this reason, in the master thesis [2] an alternative algorithm is presented that addresses all these problems: the island algorithm. It outperforms the Quadtree algorithm in both compression and the ability for a fast and simple hardware implementation.

#### REFERENCES

- [1] P. Bertels. Hardware-ontwerp van een bewegingsschatter voor video-compressie, 2004.
- [2] M. D'Haene. Het ontwerpen van entropie encoding op FPGA als onderdeel van een wavelet-gebaseerde video-encoder, 2004.
- [3] I. Doms. Het ontwerpen van temporele en spatiale wavelet-transformaties op FPGA als onderdeel van een wavelet-gebaseerde video-encoder, 2004.
- [4] John G. Cleary Ian H. Witten, Radford M. Neal. Arithmetic coding data compression. 30(6):520–540, June 1987.
- [5] A. Munteanu. *Wavelet Image Coding and Multiscale Edge Detection, Algorithms and Applications, The Quadtree Coding Approach*, pages 77–106. 2003.
- [6] A. Munteanu, J. Cornelis, G. Van der Auwera, and P. Cristea. Wavelet image compression - the quadtree coding approach. *IEEE Transactions on Information Technology in Biomedicine*, 3(3):176–185, September 1999.
- [7] M. Nelson. Arithmetic coding + statistical modeling = data compression, part 1 - arithmetic coding. *Dr. Dobb's Journal*, February 1991.
- [8] M. Nelson. Arithmetic coding + statistical modeling = data compression, part 2 - statistical modeling. *Dr. Dobb's Journal*, February 1991.
- [9] P. Schelkens, A. Munteanu, J. Barbarien, M. Galca, X. Giro-Nieto, and J. Cornelis. Wavelet coding of volumetric medical datasets. *IEEE Transactions on Medical Imaging*, September 2001.
- [10] P. Xiao. Image compression by wavelet transform. Master's thesis, East Tennessee State University, 2001.