

Scalable and Hardware-friendly Wavelet Entropy Coding

Hendrik Eeckhaut, Mark Christiaens, Harald Devos, Benjamin Schrauwen, Dirk Stroobandt
Ghent University ELIS-PARIS

St. Pietersnieuwstraat 41, 9000 Gent, Belgium
{heekhau,mchristi,hdevos,bschrau,dstr}@elis.ugent.be

Abstract—In the RESUME project we explore the use of reconfigurable hardware for the design of portable multimedia systems by developing a scalable wavelet-based video codec. A scalable video codec provides the ability to produce a smaller video stream with reduced frame rate, resolution or image quality starting from the original encoded video stream with almost no additional computation. This is important for portable devices that have different Quality of Service (QoS) requirements and power restrictions. Conventional video codecs do not possess this property; reduced quality is obtained through the arduous process of decoding the encoded video stream and recoding it at a lower quality. Producing such a smaller stream has therefore a very high computational cost.

In this article we present the results of our investigation into the hardware implementation of such a scalable video codec. We focus on the “entropy decoder” (ED), which is a significant bottleneck. The ED consists of two sub-components: At the lowest level there is an arithmetic decoder (AD) that decodes the texture bit stream. At a higher level the Model Selector (MS) characterizes the pixel environment and steers the AD. Originally the “QuadTree Limited” (QTL) algorithm [1], [4], [5], [6], [7] was used for ED. The QTL-algorithm is a state of the art wavelet image compression algorithm that is said to outperform zerotree-like encoders and inherently supports resolution and quality (PSNR) scalability. Unfortunately its complexity, large memory footprint, irregular memory access pattern, and recursive quadtree structure hamper an elegant hardware implementation.

We present an alternative, hardware-friendly algorithm for entropy coding with superior data locality (both temporal and spatial), with a small memory footprint and excellent compression while maintaining all scalability properties.

I. INTRODUCTION

“Scalable video” is encoded in such a way that it allows to easily change the Quality of Service (QoS) i.e. the frame rate, resolution, color depth and image quality of the decoded video, without having to change the video stream used by the decoder (except for skipping unnecessary blocks of data without decoding) or without having to decode the whole video stream if only a part of it is required.

Such a scalable video codec has advantages for both the server (the provider of the content) and the clients. On

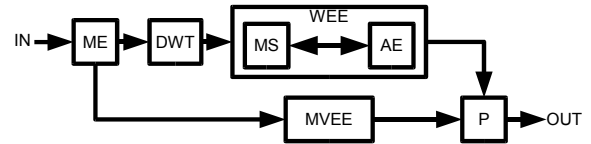


Fig. 1. High-level overview of the video encoder

the one hand the server scales well since it has to produce only one video stream that can be broadcast to all clients, irrespective of their QoS requirements. On the other hand the client can easily adapt the decoding parameters to its needs. A home cinema system can decode the stream at full quality, while a small portable client can decode the stream at low resolution and frame rate without needing the processing power of the larger clients. This way the decoder can optimize the use of the display, required processing power, required memory, ...

The internal structure of one implementation of a scalable encoder is shown in Figure 1 and was described in [1], [4], [5], [6], [7]. It consists of the following parts:

ME: “Motion Estimation” exploits the temporal redundancy in the video stream by looking for similarities between adjacent frames. To obtain temporal scalability (i.e. adjustable framerate of the video), motion is estimated in a hierarchical way as illustrated in Figure 2. This dyadic temporal decomposition enables decoding of the video stream at different bitrates. The decoder can choose up to which (temporal) level the stream is decoded. Each extra level doubles the frame rate.

An intermediate frame is predicted from its reference frames by dividing it into macroblocks and comparing each macroblock to macroblocks in the reference frames. The relative position of the macroblocks in the reference frames with respect to the intermediate frame is stored as motion vectors. The difference between the predicted and the original frame is called an “error frame”.

MVEE: “Motion Vector Entropy Encoder” is responsible for entropy encoding the motion vectors.

DWT: The “Discrete Wavelet Transform” takes a reference or error frame and separates the low-pass and high-pass components of the 2D image as illustrated in Figure 3.

Each LL-subband is a low resolution version of the original frame. The inverse wavelet transform (IDWT) in the decoder can stop at an arbitrary level, resulting in resolution scalability.

WEE: The “Wavelet Entropy Encoder” is responsible for entropy encoding the wavelet transformed frames. The frames are encoded bitplane by bitplane (from most significant to least significant), yielding progressive accuracy of the wavelet coefficients (Figure 4). The WEE itself consists of two main parts: the “Model Selector” (MS) and the “Arithmetic Encoder” (AE). The MS provides the AE with continuous guidance about what type of data is to be encoded by selecting an appropriate model for the symbol (a bit) that has to be encoded next. It exploits the correlation between neighboring coefficients in different contexts. Finally the AE performs the actual compression of the symbolstream.

P: The “Packetizer” packs all encoded parts of the video together in one bit stream representing the compressed video.

Scalability in color depth is obtained by encoding luminance and chrominance information in three different channels in the YUV 4:2:0 format. Omitting the chrominance channels yields a grayscale version of the sequence, allocating more bits to these channels increases the color depth. Motion estimation is computed from luminance information only, but is also applied to the chrominance channels. In the other parts of the algorithm the channels are processed totally independent.

By inverting the operations of Figure 1 we obtain a scalable video decoder consisting of a Depacketizer (DP), a Motion Vector Entropy Decoder (MVED), a Wavelet Entropy Decoder (WED), an Inverse Discrete Wavelet Transform (IDWT) and Motion Compensation (MC). The wavelet entropy decoder described in [1], [4], [5], [6], [7] is our focus in this paper. Since our final goal is to achieve real-time performance we need hardware acceleration. We target an FPGA implementation to effectively support scalability.

After profiling the software implementation of this decoder (Figure 5) we came to the conclusion that the real-time performance is severely limited by the Wavelet Entropy Decoder (WED). The reason for this is that the WED encodes each frame one symbol (a bit) at a time. To get a feel for the orders of magnitude: the WED must decode approximately $30 \cdot 10^6$ symbols/second for a CIF video (resolution: 352×288) playing at a framerate of 30 Hz. We found that the algorithms described in [1], [4], [5], [6], [7] have a bad spatial and temporal locality and require data structures that are too large for an efficient hardware implementation. In this paper we present an alternative

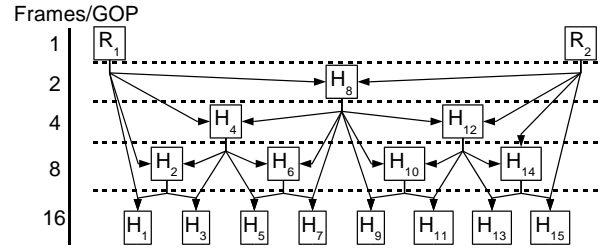


Fig. 2. Framerate scalability. Motion estimation processes one Group of Pictures (GOP) consisting of 16 consecutive frames. The arrows illustrate which frames are used as a first approximation of the intermediate frames at lower compositional levels. R_1 is the reference frame of this GOP, R_2 is the reference frame of the next GOP and the H_i are the intermediate frames.

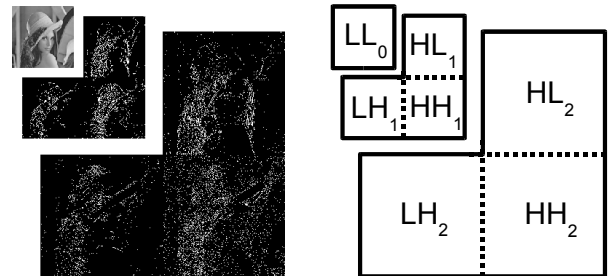


Fig. 3. Resolution scalability. Numbers in subscript reflect the resolution layers.

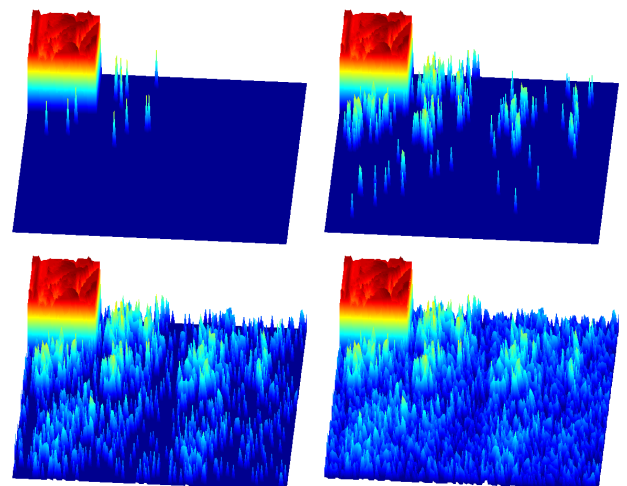


Fig. 4. Quality scalability: decoding more bitplanes gives a more accurate wavelet transformed frame.

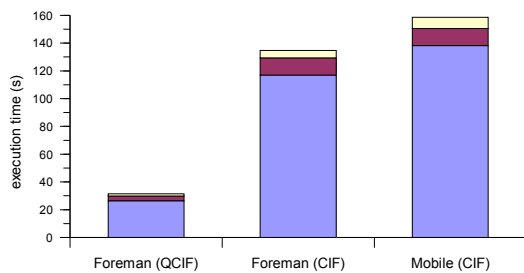


Fig. 5. Execution times for decoding three reference sequences (9.6s each) of the three major parts of the video codec. From bottom to top: Wavelet Entropy Decoding (WED), Inverse Discrete Wavelet Transform (IDWT) and Motion Compensation (MC)

algorithm that is tailored to a hardware implementation.

II. A HARDWARE-FRIENDLY WAVELET ENTROPY DECODER

We have designed a WED with the following properties in mind:

- In the first place a WED should support the scalability of the codec; this is both resolution and quality scalability. As mentioned in the Introduction, quality scalability is obtained by encoding the wavelet image bitplane by bitplane. Resolution scalability requires that data from the different resolution layers is encoded independently in the video stream. This enables us to only decode those resolution layers that are required to achieve the desired resolution.
- The algorithm should also be economical with memory. The working set should be as small as possible to avoid memory accesses to become a bottleneck.
- A high degree of parallelism is necessary if we want a really fast hardware (FPGA) implementation.
- A related issue is simplicity so as to encourage an elegant implementation.
- Finally a competitive compression rate should be achieved.

A. The Algorithm

We propose a new algorithm as shown in Figure 6. All subbands of the wavelet transformed channel are encoded (and decoded) totally independently. So it is possible to process all subbands of the wavelet transformed color channel of the frame in parallel (Figure 7). The subbands are processed bitlayer per bitlayer from top to bottom. The top is the bitplane that contains the most significant bit of the largest absolute value of all coefficients and the bottom is the bitplane containing the least significant bits. The bitlayers are processed in scanline order. This greatly ben-

```

encode_frame:
  for all subbands:
    load_subbandmodel
    if (LL-subband of reference frame)
      encode (mean of frame, data model)
      subtract mean from resolution level
      encode (number top bitplane, data model)
    for all bitplanes
      encode_bitplane

encode_bitplane:
  for all bits //scanline order
    if coefficient was not significant yet
      if starting bitplane
        encode (bit, toplayer model)
      else
        encode (bit, significance model)
    if bit becomes significant
      encode (sign, sign model)
      update bitmaps
    else //was already significant
      encode (bit, refinement model)

```

Fig. 6. Entropy encoding of one wavelet encoded frame.

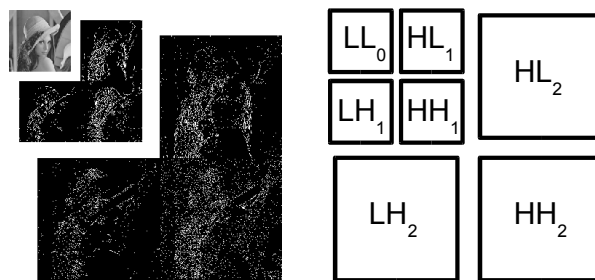


Fig. 7. All subbands can be encoded/decoded in parallel.

efits the memory accesses, since this is the order in which the data is stored in memory. It also enables us to stream data and use burst mode features of slower memories. All data from one subband is processed sequentially since all bits are now encoded based on information of previously encoded bits.

The first subband, the LL-subband of resolution layer 0, of reference frames is treated slightly differently because it contains, in contrast with all other subbands, only positive coefficients. This is a consequence of the use of the 9/7 biorthogonal filter pair in the wavelet transform. To avoid encoding the signs and to give this subband similar properties as the other subbands, the mean value of this LL-band is subtracted of all coefficients. Encoding this value first, gives the additional advantage of a good and compact approximation of all pixels, when decoding at very low bitrates.

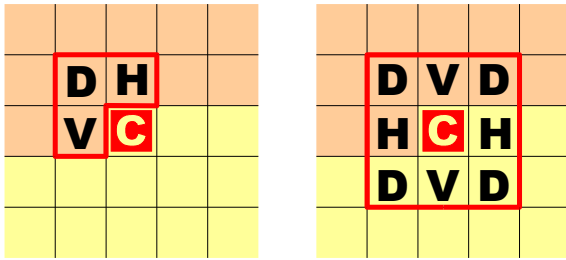


Fig. 8. Illustration of the use of the bitmaps: neighborhood of the top bitplane (left) and of the remaining bitplanes (right). ‘C’ is the current bit.

As can be seen from the code in Figure 6 symbols are encoded with different models (the second argument of the `encode` routine) depending on their context. To improve memory accesses this context is kept very small. A model contains information about the expected value of the incoming symbol and is used to encode this symbol as efficiently as possible in the Arithmetic Coder (AC).

There are four types of models:

- The `data` models are used to encode data such as the number of the starting bitplane and the mean value of the LL-subband.
- The `sign` models assist in the prediction of the signs of the wavelet coefficients.
- The `significance` models are used to predict the most significant bit of each wavelet coefficient. A coefficient is called significant as soon as we encounter its most significant bit. This group of models is subdivided further into two classes: the highest bitplane models and the remaining bitplane models. The selection of highest bitplane models depends on the significance of the surrounding pixels that have already been encoded (Figure 8). The selection of the remaining bitplane models is based on the significance of *all* surrounding pixels since all pixels are already encoded up till at least the previous bitplane.
- The `refinement` models actually consist of only one model, used to estimate the value of all refinement bits. Refinement bits are the bits following the most significant bit. These are the remaining bits we come across when processing lower bitplanes than the bitplane where the wavelet coefficient became significant. These bits have the characteristics of noise and are therefore hard to predict.

The Model Selector (see Figure 9) is responsible for selecting the models. Models are selected based on information regarding previously encoded bits. Model selection is used to exploit statistical characteristics (e.g. the fact that pixels become significant in clusters) by encoding symbols

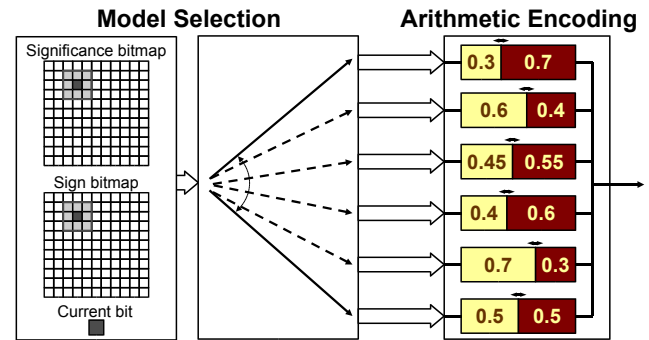


Fig. 9. The model selector selects a model based on information of neighbors, stored in sign- and significance bitmaps. Once the model is determined, the arithmetic encoder encodes the bit with the selected model.

with a similar distribution using the same arithmetic coder.

For optimal compression, storing all information about previously processed data would be ideal but since this excludes an efficient hardware implementation only the most relevant information is stored. Our algorithm limits this information to the sign and the significance of each coefficient. This information can easily be organized as two bitmaps with dimensions equal to the subband’s.

From these bitmaps the number of horizontal, vertical and diagonal significant (or negative) neighbors of the current coefficient are counted to determine the model for the arithmetic coder (Figure 8). In total there are 64 models:

- 1 `data` model.
- 27 `sign` models: To determine the sign model in the horizontal, vertical or diagonal direction, the number of negative neighbors is subtracted from the number of positive neighbors. Non-significant neighbors are not counted. Depending on the sign of this subtraction, the sign in each direction is more likely to be positive (+), negative (−) or none of both (?). Each sign model is a combination of the result in the three directions.
- 8 highest bitlayer `significance` models: one for each possible combination of the significance of three already visited neighbors. (Figure 8)
- 27 remaining bitlayers `significance` models: one for each combination of 0, 1 or 2 significant horizontal neighbors; 0, 1 or 2 significant vertical neighbors; and 0, 1 or more significant diagonal neighbors.
- 1 `refinement` model.

To determine the models of coefficients at the border of the subband, the bitmaps are extended with a symmetric expansion.

B. Arithmetic Coder

For the arithmetic coder we opted for a modified version of the CABAC arithmetic entropy encoder used in the AVC codec [2]. This is a low-complexity adaptive, binary arithmetic coder with a probability estimation algorithm that is well suited for an efficient hardware implementation.

We made a few changes to this arithmetic coder to fit it better in our wavelet entropy encoder. Since all memories on an FPGA are 9 bit wide, we augmented the 7 bit state per model (i.e. the current estimated probability of the model) to 9 bit. This increased the accuracy for probability estimation and as a consequence the compression performance. We also perfected the transition rule table for updating the probability estimation, but this falls outside the scope of this paper. The fact that only a 9 bit state per model needs to be stored, means that we only require 576 bits. The cost for a large number of models is in other words very limited.

C. Warm up of models

Arithmetic coders perform very good if they are able to accurately estimate the probability of the incoming bitstream. This is achieved by guiding the arithmetic coder with models, that in the ideal case stand for a certain fixed probability, resulting in near optimal compression. But since we are using a high number of models, how can the arithmetic coder estimate the probability of the models that are rarely used? We tackled this problem by estimating the probabilities beforehand, by observing the real probabilities for a set of reference video sequences. By initializing each model with these precalculated values we reach the actual probability much sooner than if we initialized the model conservatively at 0.5.

D. Subband models

There are a lot of different types of subbands which all have distinct statistical properties. In the first place there are large differences between the LL, HL, LH and HH subbands. In addition, models will be different for subbands of different resolution layers. If we also take the difference between the color channels and the position in the temporal frame hierarchy into account, we distinguish 480 different types of subband models (for 4 resolution levels). Each type has its own private 64 arithmetic models. Since all we have to do when coding a certain subband is swap in the appropriate subband model that initializes the 64 arithmetic coder states, no real efforts were made to reduce this high number of different subband models; this cost is negligible.

TABLE I
CONFIGURATION OF ALTERA STRATIX AND XILINX VIRTEX II PRO ON-CHIP MEMORY

Altera Stratix Blocks			Virtex II Pro
M512-RAM	M4K-RAM	M-RAM	Select-RAM
512×1	$4K \times 1$	$64K \times 8$	$16K \times 1$
256×2	$2K \times 2$	$64K \times 9$	$8K \times 2$
128×4	$1K \times 4$	$32K \times 16$	$4K \times 4$
64×8	512×8	$32K \times 18$	$2K \times 8$
64×9	512×9	$16K \times 32$	$2K \times 9$
32×16	256×16	$16K \times 36$	$1K \times 18$
32×18	256×18	$8K \times 64$	512×36
	128×32	$8K \times 72$	
	128×36	$4K \times 128$	
		$4K \times 144$	

III. MEMORY USE

In Section II we stated that the codec had to be really economical with memory since bandwidth is often a bottleneck for multimedia applications. Randomly accessing large data structures on an FPGA is not recommended since memory resources are limited and external memories might not be fast enough. Since the decoder has to be at least real time, it is very important that the working set, the data structures that are accessed very frequently, are small enough to fit in small/fast on-chip memory.

To have a better idea what the types of available memory are, Table I shows the available on-chip memory blocks of resp. the Altera Stratix and the Xilinx Virtex II Pro of Xilinx. This on-chip memory is dual-port and has a latency of 1 clock cycle. A typical Altera Stratix (EP1S25) has 224 M512 (64×9 bit) blocks, 138 M4K (512×9 bit) blocks and two MRAM ($64K \times 9$ bit) blocks. Similarly a typical Xilinx Virtex II Pro (XC2VP30) has 136 Select-RAM ($2K \times 9$ bit) blocks.

Implementing the above WED algorithm in a Stratix EP1S25 or Virtex XC2VP30 requires the following memory blocks. For the AD we require:

- A lookup table for determining the next state to jump to after processing a symbol. There are 512 such states but the table is symmetric so we are dealing with a table of size $512/2 \cdot \log_2(512/2)$ bits = 256 bytes. This will typically fit in one M4K or one Select-RAM block.
- A lookup table containing the current state of 64 models. One such state requires 9 bits. This is very convenient since almost every FPGA has on-chip RAM that can be addressed 9 bits at a time. This will fit in one M-512 block or one Select-RAM block.
- A lookup table used to look up the new size of the range

being coded (see [3]). This lookup table consists of 256 entries (for each of the symmetrical states). Each entry contains a value for 4 quantization levels that needs 16 bits resulting in a total size of this table of 2048 bytes. Depending on the chosen FPGA component this will require 4 M4K blocks or 1 Select-RAM block.

- A small buffer to store a part of the bit stream that is being decoded will be necessary. One M4K block or one Select-RAM block should suffice.

The MS has larger, but still very modest, memory requirements:

- The sign bitmap for the subband under reconstruction. For a CIF image this data structure is at most 25344 bits. To implement this we need 6 M4K blocks or 2 Select-RAM blocks.
- An identical data structure is necessary for storing the significance bitmap.
- We need a data structure to store the decoded image. The WED algorithm is constructed so that it reconstructs images one bitplane at a time and in Z-order. This means we can stream out one bitplane at a time requiring only a small buffer.

In total we find that one WED will require approximately 20 M4K blocks (14.5% of the M4K blocks of an EP1S25) or 9 Select-RAM blocks (6.6% of the XC2VP30).

An alternative layout for the Stratix platform is to use M-RAM blocks together with M4K blocks. We can fit the whole significance and sign bitmaps into one M-RAM block. The remaining data structures stay in the M4K blocks. We would need two buffers for the bitmaps as they are being streamed from the M-RAM block. Assuming that we can fit the bitmaps of additional WEDs into the same M-RAM block, we would be using only 8.6% of one M-RAM block of an EP1S25 per subband decoder. The remaining 10 M4K blocks form only 7.25% of the M4K blocks.

IV. RESULTS

In Figure 10 the compression ratio of the new Wavelet Entropy Encoder was compared with the QTL approach of [5], [6] for 289 frames of the “mobile” sequence (CIF, 30 frames/s, full resolution). For lossless decoding, the new approach is about 2% better. Due to the fact that WEE needs to encode the number of the highest bitplane (HB) for each subband instead of for each resolution layer, the fixed cost (decoding zero bitplanes, thus only the HB) is about twice as large (after compression) as QTL. To illustrate the compression without this effect, we also measured the relative compression of the WEE when we used only one HB per resolution layer. Now the WEE outperforms

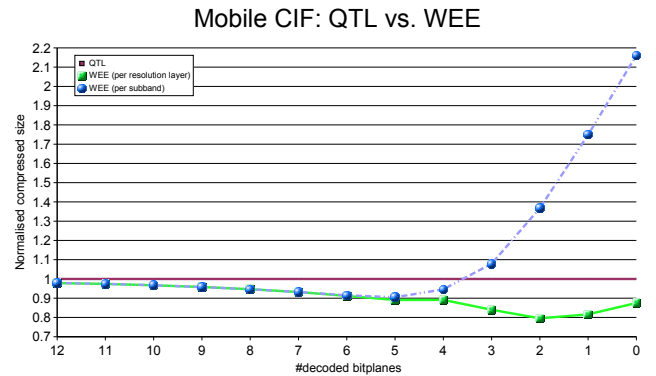


Fig. 10. Compression ratio of WEE compared with QTL for a decreasing number of decoded bitplanes. To variants of WEE are plotted (per resolution and per subband) to illustrate the relative cost of encoding the number of the highest bitplane.

the QTL for all bitplanes. But since the absolute value of this fixed cost is very small, we preferred the added parallelism above a slightly better compression (at very low bitrates).

In Figure 11, the average PSNR for decoding 49 CIF and QCIF frames from the “mobile”, “foreman” and “silent” sequences (30 frames/s, full resolution) are plotted for both the QTL and the WEE approach. The PSNR of each frame is calculated as follows. If $Y_{i,j}$, $U_{i,j}$ and $V_{i,j}$ and $Y'_{i,j}$, $U'_{i,j}$ and $V'_{i,j}$ are resp. the luminance and the two chrominance channels of the original and reconstructed frame of $h \times w$ pixels, then the PSNR is defined as follows:

$$10 \log_{10} \frac{255^2 \frac{3}{2} hw}{\sum (Y - Y')^2 + \sum (U - U')^2 + \sum (V - V')^2} \quad (1)$$

Both approaches used the same motion estimation and wavelet transform, only the entropy encoding and bitrate allocation algorithm were different. The bitrate allocation algorithm of WEE is a quick hack of QTL’s algorithm and less elaborated. Moreover QTL processes each bitplane in three passes to have a finer arrangement of the most quality enhancing bits. This allows a better bitrate allocation but slows down entropy decoding. Therefore WEE processes each bitplane in one pass.

Figure 11 clearly illustrates that although a lot of concessions were made the new WEE is at par with QTL, which is considered state of the art [4].

V. CONCLUSIONS

We presented a balanced algorithm for scalable wavelet entropy coding with superior temporal and spatial data locality, a small memory footprint, streaming capability, a high degree of parallelism and an excellent compression.

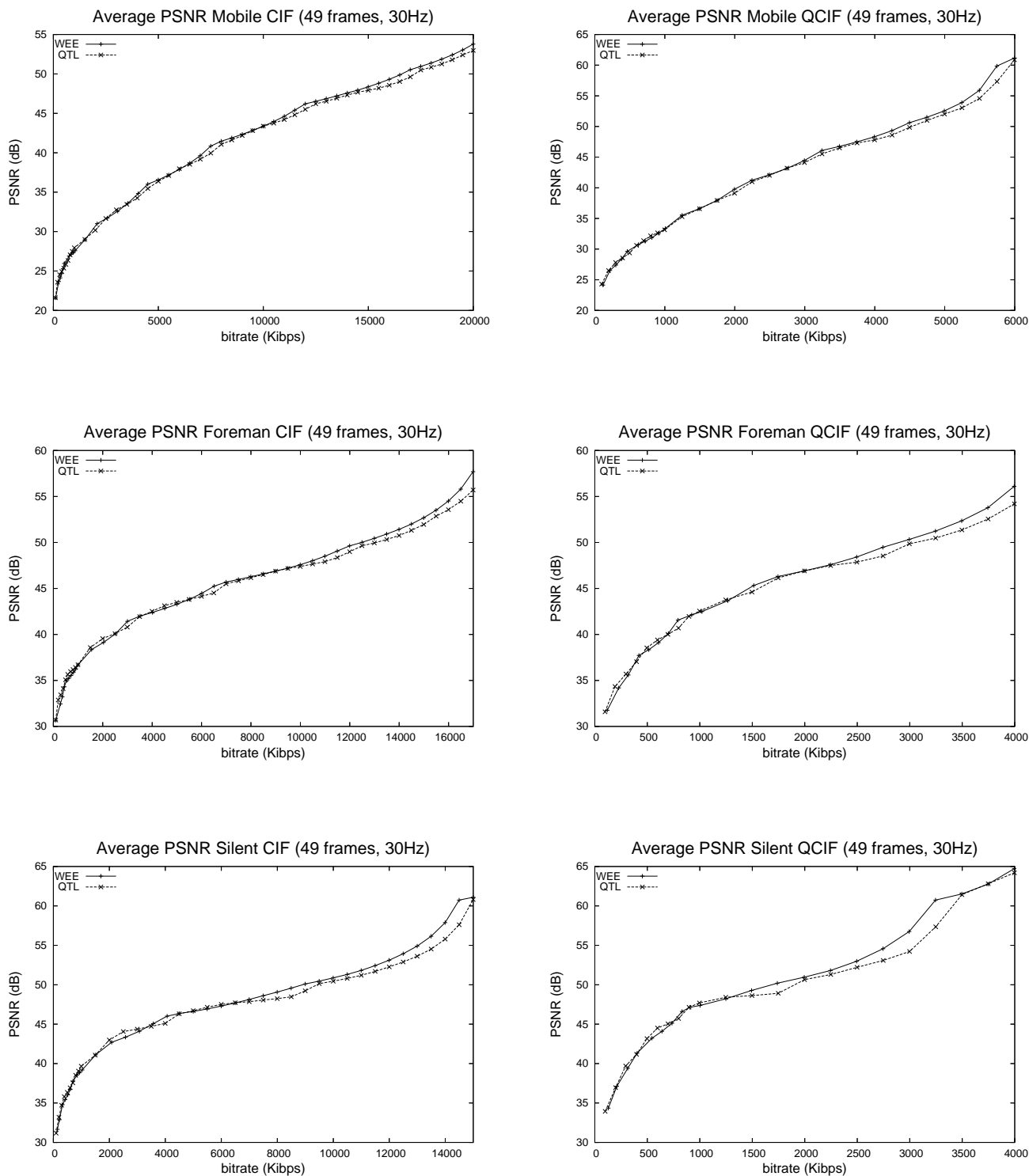


Fig. 11. Results for decoding 49 CIF and QCIF frames from the “mobile”, “foreman” and “silent” sequences.

VI. ACKNOWLEDGEMENTS

This research is supported by I.W.T. grant 020174, F.W.O. grant G.0021.03 and by GOA project 12.51B.02 of Ghent University. Harald Devos is supported by the fund for scientific research Flanders (F.W.O.).

REFERENCES

- [1] H. Devos, H. Eeckhaut, M. Christiaens, F. Verdicchio, D. Stroobandt, and P. Schelkens. Performance requirements for reconfigurable hardware for a scalable wavelet video decoder. In *CD-ROM Proceedings of the ProRISC / IEEE Benelux Workshop on Circuits, Systems and Signal Processing*. STW, Utrecht, November 2003.
- [2] D. Marpe, H. Schwarz, G. Blättermann, G. Heising, and T. Wiegand. Context-based adaptive binary arithmetic coding in JVT/H.26L. *Proc. IEEE International Conference on Image Processing (ICIP'02)*, 2:513–516, September 2002.
- [3] D. Marpe, H. Schwarz, and T. Wiegand. Context-based adaptive binary arithmetic coding in the H.264 / AVC video compression standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):620–636, July 2003.
- [4] A. Munteanu. *Wavelet Image Coding and Multiscale Edge Detection - Algorithms and Applications*. PhD thesis, Vrije Universiteit Brussel, 2003.
- [5] 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.
- [6] 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, Special Issue on "Wavelets in Medical Imaging"*, 22(3):441–458, March 2003.
- [7] D. Stroobandt, H. Eeckhaut, H. Devos, M. Christiaens, F. Verdicchio, and P. Schelkens. Reconfigurable hardware for a scalable wavelet video decoder and its performance requirements. *Computer Systems: Architectures, Modeling, and Simulation*, 3133:203–212, July 2004.