SBHP - A LOW COMPLEXITY WAVELET CODER

Christos Chrysafis, Amir Said, Alex Drukarev
Hewlett Packard Laboratories
1501 Page Mill Road, MS 3U-3
Palo Alto CA 94304-1126

Asad Islam, William A. Pearlman
Electrical, Computer and Systems Engineering Dept.
Rensselaer Polytechnic Institute
Troy, NY 12180

ABSTRACT

We present a low-complexity entropy coder originally designed to work in the JPEG2000 image compression standard framework. The algorithm is meant for embedded and non-embedded coding of wavelet coefficients inside a subband, and is called *Subband-Block Hierarchical Partitioning* (SBHP). It was extensively tested following the standard experiment procedures, and it was shown to yield a significant reduction in the complexity of entropy coding, with a small loss in compression performance. Furthermore, it is able to seamlessly support all JPEG2000 features. We present a description of the algorithm, an analysis of its complexity, and a summary of the results obtained after its integration into the Verification Model (VM).

Introduction

One important trend occurring in the whole digital imaging industry is the increase in image size and resolution. It is a consequence of the development of better and less expensive image acquisition devices. This trend is certain to continue, because digital imaging can only replace other technologies by providing superior resolution and quality.

Baseline JPEG is a low-complexity solution, but it is not able to support all the features planned for JPEG2000. Supporting a rich set of features often leads to an increased complexity of a compression system. In view of this it would be interesting to explore low complexity algorithms for such problems. We want to know how much faster they can be compared with the scheme chosen for the JPEG2000 standard, and measure the loss in compression efficiency due to approximations and simplified coding. At the same time, we have to be very careful to conduct a fair comparison, i.e., we want to compare algorithms that support the same features, and test them in exactly the same conditions.

From a functional point of view, SBHP does exactly the same tasks executed by the entropy coding routines used on the JPEG 2000 Verification Model (VM) [1]. In consequence, every single feature and mode of operation supported by the VM continues to be available with SBHP. SBHP operates in the EBCOT [2] framework chosen for the JPEG2000 standard. Like EBCOT and other encoders, SBHP is applied to blocks of wavelet coefficients extracted from inside subbands. It produces a fully embedded bit stream that is suitable for several forms of progressive transmission, and for one-pass rate control. Except for the fact that it does not use the arithmetic encoder, it does not require any change in any of the VM functions outside entropy coding.

The SBHP performance was measured after its integration into VM 4.2. It was found that the SBHP encoder runs about 4 times faster, and the decoder is about 6 to 8 times faster, depending on the platform.

Design Objectives

The basic concepts behind SBHP can be found in the references [3-10]. Here we present a brief description of the coding algorithm, plus the implementations details. Some other topics are discussed in order to explain why SBHP has such low complexity. To do this, we discuss some basic facts about practical complexity of entropy coding.

Some important properties that need to be exploited to minimize the complexity during bit-plane entropy coding are:

1. Typically, blocks of bits with high probability of being zero cluster together. Coding them in blocks is much more efficient than coding one-by-one.
2. The size of the blocks of zeros is by itself a context for entropy coding, which is much less costly to compute than aggregating contexts from causal neighborhoods.
3. Some bits have probability very near ½. Trying to compress them is a waste of computational resources.
4. Entropy coding methods do add to complexity, and adaptive arithmetic or adaptive Huffman coding add much more.
5. Coding techniques can adapt to different distributions of wavelet coefficients without costly management of statistics and contexts, and without having many codebooks.

In order to achieve minimal complexity SBHP was designed to exploit all these properties. It does not use arithmetic coding, and uses only two fixed 15-symbol Huffman codes in some special conditions. Although arithmetic coding is a powerful and versatile tool, it should be clear that even the fastest arithmetic coding implementations are much slower than, for example, just moving "raw" bits from the wavelet coefficient to the bit stream.

Coding Algorithm

SBHP is an implementation of a coding method that was proposed for JPEG2000, called SPECK [3,5,9]. It shares some features of NQS [4] (same type of data), but it is a different coding method. Like those two methods, it is based on a set-partitioning strategy [3-10], which is explained below. This type of coding was shown to produce excellent compression [3] with low complexity, and even without any further form of entropy coding [7].

The first implementations of set partitioning were specific to the hierarchical wavelet decomposition, working across subbands. For that reason, it was thought that such algorithms could only work for that type of data. The numerical results shown in this report and

[4,5] prove that this is not true. SBHP is efficient without exploiting inter-subband relationships.

SBHP uses the standard form of bit-plane coding [3,4,5,6] to create an embedded bit stream. Two main problems have to be addressed for efficient coding of bit planes:

1. Large areas of the bit plane have bits equal to zero. They have to be compressed in an efficient and fast manner.
2. Pixels have to be visited in an order optimizing the rate-distortion properties of the embedded bit stream.

The coding algorithm in VM4.2 solves the first problem using context-based arithmetic coding (with a quite sophisticated context creation and management) [2], and solves the second problem using multiple passes per bit plane.

SBHP solves the same problems in quite different ways, employing the concepts of set partitioning and using order lists. It requires only one coding pass per bit plane, and the number of pixels tested is normally much smaller than the number of pixels in a block.

Set Partitioning

Consider an image that has been adequately transformed using an appropriate subband transformation (such as the discrete wavelet transform). The image transform C is represented by an indexed set of transformed coefficients $c_{i,j}$. Following the ideas in [7], for a given bit plane n and a given set B of coefficients (now called pixels for simplicity), we define the significance function:

$$S_n(B) = \begin{cases} 1, & \text{if } \left(\max_{(i,j)\in B} |c_{ij}|\right) \geq 2^n, \\ 0, & \text{otherwise.} \end{cases}$$

Following this definition, we say that set B is *significant* with respect to bit plane n if $S_n(B) = 1$. Otherwise, we say that set B is *insignificant.*

SBHP codes only two types of data: significance data (i.e., values of $S_n(B)$) and bits extracted directly from the wavelet coefficient binary representation (i.e., "raw" bits from the bit planes). The set-partitioning process used by SBHP is almost the same used by SPECK [3]. Figure 1 shows the sequential process of splitting a 16×16 block of wavelet coefficients, and below we explain exactly how the process is done. First, let us define the conventions used in Figure 1. The continuous lines represent the boundaries of pixel sets. The dashed and dotted lines indicate the boundaries of the subsets in which a set is to be split. After reaching the level of individual pixels, those pixels are shown as a gray area.

The algorithm starts with two sets, as shown in Figure 1 (a). One composed of the 2×2 top-left pixels in the block, and the other is the set of remaining pixels. The first set can be decomposed in 4 individual pixels, and the second can be decomposed in three 2×2 blocks and the remaining pixels. Figure 1 (b) shows the next level of decomposition: each 2×2 set can be decomposed in 4 pixels, and the remaining set can be partitioned into three groups of 4×4, plus the remaining pixels. In the next stage, each 4×4 set is split into 4 2×2, and the remaining set is partitioned in 8×8. At this moment there is no "set of remaining pixels," and the decomposition occurs only in a standard quadtree manner. Figure 1 (d) shows how the process continues, until all sets are partitioned to individual pixels.

Note that Figure 1 just shows the rules to partition each set. During coding the different sets are partitioned at different times, according to the pixel values. This is detailed in the next section. The only difference between the scheme shown above and SPECK, is that SPECK starts with a single pixel set, while SBHP always has sets with at least 2×2 pixels.

Coding Sequence

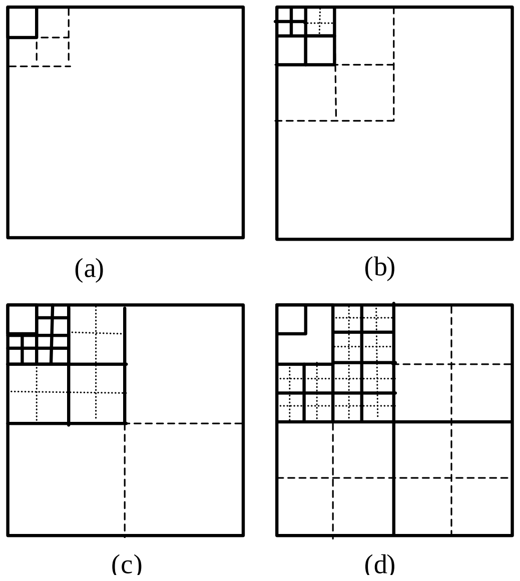

Figure 1 Set partitioning rules used by SBHP.

Like other set-partitioning schemes [3,4,7], SBHP uses three lists to minimize the number of tests for a given bit-plane.

- LIS (List of Insignificant Sets) – all the sets (with more than one pixel) that are insignificant but do not belong to a larger insignificant set.
- LIP (List of Insignificant Pixels) – pixels that are insignificant and do not belong to insignificant sets.
- LSP (List of Significant Pixels) – all pixels found to be significant in previous passes.

For each new bit plane we update the lists. The list entries are visited in the following order

1. Test significance of pixels in the LIP.
2. Test significance of sets in the LIS (including new subsets generated at the same pass).
3. Code refinement bits for pixels in the LSP, except those pixels that were added to the LSP in same bit plane.

When a set is split, the probability that a generated subset is significant is smaller than ½. This fact is exploited to reduce the number of compressed bits with simple entropy coding. Since there are four subsets or pixels, we can code them together. We have chosen a Huffman code with 15 symbols, corresponding to all the possible outcomes. (A set is split when at least one of the subset is significant, so not all subsets can be insignificant after splitting.)

No type of entropy coding is used to code the sign and the refinement bits. Of course, this results in compression loss, but it is observed that it is very hard to compress these bits efficiently, and nothing is simpler than just moving those "raw" bits to the compressed stream. To optimize the rate-distortion properties of the embedded bit stream we sort the elements in the LIS, LIP and LSP.

- LSP and LIP: pixels added first are coded first (FIFO).
- LSP: sets with smallest number of elements are processed first. When sets have the same number of elements, those added first are coded first.

The FIFO is actually the most efficient for list management. Sorting the sets by size needs some more thought. A good solution is to use several lists [9]: this keeps the sets sorted with very low management cost.

Rate-Distortion Optimization

The one-pass bit-rate control in VM 4.2 requires sorting the data according to its rate distortion properties. The coding function returns the increase in bit rate and the decrease in distortion for each bit-plane coding pass. The computation of the number of bits is a trivial matter, but the computation of decrease in distortion is not. Exact computation of the squared-error would require computation of square values for each coded pixel.

SBHP can use two properties to simplify this computation. First, the derivative of the rate-distortion function can be predicted with high precision at the beginning of each refinement pass (where exactly one bit is used for each pixel and the distortion reduction pretty much follows a simple statistical model). The second property comes from the fact that each coding action corresponds to a list entry. It happens that the average reduction in distortion can be reliably estimated as a function of the number of elements in the LIS, LIP and LSP.

For each bit plane, there are three rate-distortion points. The first point includes all new significant bits in LIP, the second point includes all new significant bits in LIS, while the third includes all new refinement bits. These three points can be seen as a way to achieve the same effect as the fractional bit plane coding used in the VM without having to go through several passes through the bit plane. In our implementation, we can chop off the bit stream in arbitrary positions, which is particularly valuable when we want to achieve exact rate control in small images.

Complexity Analysis

The SBHP encoder first visits all pixels to gather information about bits in all bit planes (*preprocess pass*). This pass is actually quite simple, requiring one bitwise OR operation per pixel, following a predetermined sequence, and some analysis of partial results. All other bit-plane coding algorithms must compute the same data to determine the first significant bit plane with at least one non-zero bit.

The set-partitioning process uses exactly one bit to indicate when all bits in a group inside a bit plane are equal to zero. The information about these groups is gathered in the preprocess pass, so only one comparison is required (the decoder just reads the bit) per bit in the *compressed stream.* This property can be used to easily show how SBHP minimizes the coding complexity and how it is asymptotically optimal.

We can roughly measure the complexity of coding a bit plane by counting the number of bit comparisons (equal to 0 or 1?) used to test the bits in the bit plane. A direct method of compression needs to visit all pixels, so its complexity is proportional to the number of pixels (multiple passes may increase complexity proportionally). SBHP, on the other hand, tests only the elements in its lists. The absolute minimum number of comparisons is unknown, but can be computed by the entropy of the bit plane. Since SBHP uses one compressed bit per comparison, and the number of bits generated per bit plane is equal to the number of comparisons, we can conclude that its number of comparisons is very near the optimal (or we would not have good compression).

Only the most basic operations, like memory access, bit shifts, additions, and comparisons are required by the encoder/decoder. No multiplication or division is required (even in approximate form), simplifying a hardware implementations.

Code profiling has shown that the computational effort is well distributed among the tasks of data access, list management, and writing raw bits to the compressed stream.

The fastest possible hardware and software implementations are achieved with the non-embedded mode of the coder [10]. In this way, there is no need for multiple passes within a block. This form of coding can run in approximately the same time as baseline JPEG, around 11 times faster than the VM4.2 on the decoder side. One good reason for that is that the largest Huffman code is of length 6 bits and we can use lookup table to decode instead of binary trees.

The decoder is usually faster than the encoder. The encoder always needs to visit all pixels in a block, unlike the decoder, which can skip over large blocks of zero coefficients.

The complexity analysis of SBHP can be divided in two parts: dependent and independent of the bit rate. The last one is related to the time to preprocess a block before encoding or decoding, and is not really related to entropy coding. The encoder needs one pass to identify the maximum magnitude values of all sets. Each pixel has to be visited only once. A bitwise OR operation is necessary for each pixel, and for each set. The number of sets in SBHP is 1/3 the number of pixels, so we need about 4/3 accesses per pixel. (All bit-plane coders need a similar pass to identify the top bit-plane.)

The key point for evaluating the coding complexity of SBHP is the fact that all time-related complexity measures for the algorithm, like number of operations, clock cycles, memory access, etc., are proportional to the number of compressed bits. Our experiment shows that this is indeed a good approximation.

Three facts are important in the list-management complexity analysis. First, the algorithm works with small blocks, so the list memory can be assigned in advance, and no time-consuming memory control is required. Second, the lists are updated in FIFO mode, so they are stored simply as arrays. Third, the lists grow exponentially for each bit-plane pass, and for all bit rates the complexity is mostly determined by the last pass. In other words, even if coding requires several passes, the complexity is typically less than a two-pass (per block) algorithm (and much less than several passes per-bit plane).

It is easy to evaluate the complexity of testing elements in the LSP: for each entry, a bit of a wavelet coefficient is moved to the compressed bit stream. There is no need to move elements in or out of the list. Processing the elements in the LIP is not much more complex. If a magnitude bit (significance) is zero, then the entry stays in the LIP. Otherwise, the coefficient sign is written, and the entry moves to the LSP.

	Decrease in PSNR		Increase in bit rate	
Rate (b/p)	5x3	9x7	5x3	9x7
0.0625	-0.28 db	-0.33 db	5.85%	8.08%
0.125	-0.32 db	-0.36 db	5.89%	7.71%
0.25	-0.36 db	-0.41 db	5.64%	7.06%
0.5	-0.41 db	-0.46 db	5.29%	6.45%
1	-0.41 db	-0.46 db	4.37%	5.39%
2	-0.37 db	-0.43 db	3.12%	3.90%
Lossless			1.13%	

Table 1 Decrease in PSNR and increase in bit rate by using SBHP, compared to JPEG2000 VM4.2. We present results for two different filter banks, the Daubechies 9/7 and the integer coefficient wavelet 5/3. The block size for encoding was 32x32 in all cases. The numbers are averages over 7 images, (aerial2, bike, cafe, woman, gold, hotel, txtr2).

For the most common bit rates, most of the computational effort is spent processing the LIS. Here there is a more pronounced asymmetry depending on the significance test. If a coefficient is insignificant, then it is just left on the LIS. Otherwise, it has to be partitioned into four subsets, with additional operations. If the set is not decomposed into individual pixels, then only new set entries have to be added to the LIS. The decomposition in individual pixels may need coding the sign bit.

Test Results

The SBHP implementation, in the embedded version is substantially faster than VM 4.2. On the encoder side it is around *4 times faster* on both PA-RISC processor and Pentium-II processor. On the decoder side it is around *6 times faster* on PA-RISC platform and around *8 times faster* on the Pentium-II processor. Of course those numbers vary depending on the image and the bit rate. In the *non-embedded* version of the algorithm the decoder can be as much as *11 times faster* on the Pentium-II, in which case the complexity of SBHP becomes very close to that of baseline JPEG.

We provide experimental results comparing the performance of the current coder with respect to the verification model (VM4.2) for JPEG2000 for 7 different images, (aerial2, bike, cafe, woman, gold, hotel, txtr2) from the JPEG2000 test set. We use the bit allocation algorithm described above to truncate the bit stream to the desired rate after compression. The results can be seen in Table 1. For other type of images such as compound documents the results may not be as good. But for compound documents JPEG2000 is not a good choice, approaches such as the one in [12] based on segmentations are needed.

Analysis of the experimental results shows that for most images, such as photographic, medical, etc., the SBHP PSNR is only about 0.4-0.5 dB below the VM. As an alternative way to compare the compression ratio loss, we measure SBHP file sizes compared to the VM, for the same quality. The numbers show that SBHP looses only 5–10% in bit rate for lossy compression and, only 1–2% for lossless compression for photographic images.

The results are more favorable when SBHP is used in conjunction with simple filters (5×3). In this case the average loss with respect to VM4.2, which varies with the bit rate, is no more than 0.5dB. Interestingly enough this configuration leads to the lowest possible complexity both in terms of memory and in terms of numerical operations. The image *Bike* gives the worse results with *aerial2* giving the best results. For lossless compression, there is an average loss anywhere from 1% to 2%.

Currently JPEG2000 is testing new versions of the verification model, our preliminary investigations indicate that the results in this paper still hold.

References

[1] C. Christopoulos, *JPEG2000 Verification Model 4.1,* ISO / IEC / JTC1 / SC29 WG1N1286.

[2] D. Taubman, *Embedded, independent block-based coding of subband data,* ISO / IEC JTC 1 / SC29 WG1N871.

[3] A. Islam and W. A. Pearlman, *Set Partitioned Sub-Block Coding (SPECK),* ISO / IEC / JTC1 / SC29 WG1N1188.

[4] J. Spring, J. Andrew, and F. Chebil, *Nested Quadratic Splitting*, document ISO / IEC / JTC1 / SC29 WG1N1191.

[5] W. A. Pearlman, *Performance of Set Partitioned Sub-Block Coding (SPECK),* ISO / IEC / JTC1 / SC29 WG1N1245.

[6] J.M. Shapiro, “Embedded image coding using zerotrees of wavelet coefficients,” *IEEE Trans. Image Processing*, vol. 41, pp. 3445–3462, Dec. 1993.

[7] A. Said and W. A. Pearlman, “A new, fast and efficient image codec based on set-partitioning in hierarchical trees,” *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 6, pp. 243–250, June1996.

[8] J. Andrew, “A simple and efficient hierarchical image coder,” *IEEE Int. Conf. on Image Proc.*, vol. 3, pp. 658–661, Oct. 1997.

[9] A. Islam and W.A. Pearlman, “An embedded and efficient low-complexity hierarchical image coder,” *SPIE Conf. on Visual Communications and Image Processing*, San Jose, CA, Jan. 1999.

[10] C. Chrysafis, A. Said, A Drukarev, W.A Pearlman, A. Islam, F. Wheeler. "Low complexity entropy coding with set partitioning", ISO / IEC / JTC1 / SC29 WG1N1313.

[11] J. Li and S. Lei, “An embedded still image coder with rate-distortion optimization,” *Proc. SPIE Symp. Visual Communications and Image Processing*, vol. 3309, pp. 36–47, San Jose, CA, Jan. 1998.

[12] A. Said, A. Drukarev, "Simplified Segmentation for Compound Image compression." *IEEE Int. Conf. on Image Proc*, Kobe, Japan, Oct 1999