HEWLETT® PACKARD

Memory Efficient Scalable Line-based Image Coding

Erik Ordentlich, David Taubman*, Marcelo Weinberger,
Gadiel Seroussi, Michael Marcellin[†]
Computer Systems Laboratory
HP Laboratories Palo Alto
HPL-1999-1
January, 1999

E-mail: [eor,taubman,marcelo,seroussi]@hpl.hp.com

embedded image compression,
wavelet transform,
skew coder,
Golomb coder,
low memory,
bit-plane coding

We study the problem of memory-efficient scalable image compression and investigate some tradeoffs in the complexity vs. coding efficiency space. The focus is on a low-complexity algorithm centered around the use of sub-bit-planes, scan-causal modeling, and a simplified arithmetic coder. This algorithm approaches the lowest possible memory usage for scalable wavelet-based image compression and demonstrates that the generation of a scalable bit-stream is not incompatible with a low-memory architecture.

*University of New South Wales, Sydney 2052, Australia
[†]University of Arizona, Tucson, Arizona

Memory Efficient Scalable Line-based Image Coding

Erik Ordentlich, David Taubman[1]*, Marcelo Weinberger, and Gadiel Seroussi*
Hewlett-Packard Laboratories, Palo Alto, CA 94304
Michael W. Marcellin
University of Arizona, Tucson, AZ 85721

Abstract. We study the problem of memory-efficient scalable image compression and investigate some tradeoffs in the complexity vs. coding efficiency space. The focus is on a low-complexity algorithm centered around the use of sub-bit-planes, scan-causal modeling, and a simplified arithmetic coder. This algorithm approaches the lowest possible memory usage for scalable wavelet-based image compression and demonstrates that the generation of a scalable bit-stream is not incompatible with a low-memory architecture.

1 Introduction

In low-memory image compression applications, such as printing and scanning of high-resolution images, storing the whole image in system memory is not practical. Image data arrives at the input of the encoder and is consumed at the output of the decoder one scan-line (or a few) at a time. In this paper, we are interested in image compression algorithms that permit exceptionally low complexity and very high throughput implementations in this setting, while achieving acceptable compression efficiency and generating bit-streams that are resolution and quality (or SNR–signal to noise ratio) scalable.

In order to arrive at a more concrete notion of complexity, we adopt the following simple hardware model for the compressor and decompressor. The computational resources are assumed to consist of three components: a processing/logic unit (the "chip"), fast but expensive internal ("on-chip") memory, and relatively slow, cheaper external ("off-chip") memory. The key parameters of this model, that serve to quantify overall system complexity and throughput, are the sizes of the two types of memory stores, the memory bandwidth or the number of memory accesses to each type of memory measured in bytes per image pixel, and the computational complexity (logic, throughput) of the processing unit.

We can then assess the relative merits of image compression algorithms by augmenting the traditional criteria of compression efficiency and bit-stream features and functionality (with the latter criterion emerging from the advent of embedded wavelet-based compression technology) with the above complexity criteria. Algorithms that achieve extreme points in this performance/complexity space are attractive from the stand-point that they are guaranteed to be optimal according to some application-dependent weighting of these criteria.

In this work we present an image compression algorithm that yields a resolution and SNR scalable bit-stream and achieves an extreme point in terms of memory usage, computational complexity, and latency, for the provided functionality. Yet compression efficiency is still

[1] Current affiliation is University of New South Wales, Sydney 2052, Australia.

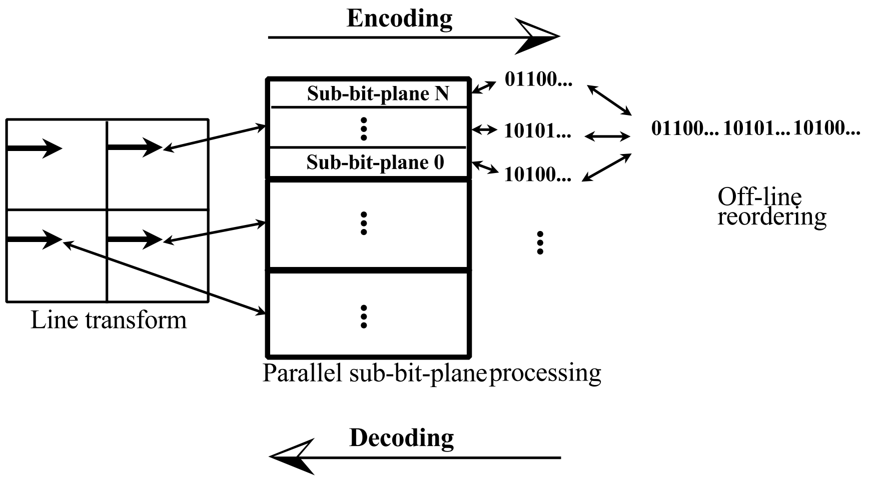

Figure 1: Line-based transform and coding architecture.

competitive with state-of-the-art algorithms. Simple variations of the basic algorithm allow for great flexibility in reducing memory bandwidth at the expense of increased memory size.

One major objective of this work is to demonstrate that the generation of a scalable bit-stream is not incompatible with a low-memory approach. A key element of the SNR scalability of the proposed algorithm is the use of sub-bit-planes, which arise from a context dependent deinterleaving of bit-planes into subsequences (sub-bit-planes), as introduced in [1] as part of an *embedding principle*. A proper ordering of the descriptions of the deinterleaved sub-bit-planes in the bit-stream yields a finer SNR scalability than does straightforward bit-plane coding. The resulting distortion-rate curve obtained by truncating an embedded bit-stream is significantly less scalloped, resulting in bit-rate dependent PSNR improvements of up to .4-.5 dB. We refer to [1] for more on the theory and motivation behind the use of sub-bit-planes.

Scan-causal modeling and simplified arithmetic coding are other key elements of the low-memory, low-complexity package that we present, and these will be discussed in subsequent sections.

2 Scalable line-based coding architecture

Figure 1 illustrates the "line-based" architecture underlying the proposed low-memory image compression algorithm. A low-memory line-based wavelet transform implementation [2] generates at the encoder K lines of transform coefficients of the four bands in a given level of the wavelet decomposition as soon as a sufficient number of lines of the low-low band of the previous decomposition level (one level closer to the original image) have been generated. At the decoder, the inverse transform consumes a corresponding number of lines of transform coefficients from each of the four bands at a given level of decomposition to generate new lines in the low-low frequency band of the next decomposition level (again, one level closer to the image). At the encoder, the memory requirements of the transform arise solely from the need to buffer up the scan lines of pixels in the image and the lines of coefficients in each low-low band of the decomposition that are necessary for computing K new lines in each band of the next coarsest decomposition level. The new coefficients that are generated in the high-pass bands pass directly into the coding engine. At the decoder, a sufficient number of lines of decoded transform coefficients must be buffered in each band of each decomposition level to compute K new lines in the low-low band of the next finer level of decomposition. In this paper we take $K = 1$ to characterize the lowest memory solution. Depending on the application and on the choice of transform, memory bandwidth considerations may favor choosing K to be as large as 4 or 8.

The coding engine consists of a collection of processing slices that encode/decode all of the sub-bit-planes of freshly generated/requested transform coefficients simultaneously. The bit-stream segments that are generated in parallel by the encoder for each sub-bit-plane are concatenated in a post-compression reordering step, to form a single embedded bit-stream. The resulting bit-stream is augmented with auxiliary information (pointers or bit-stream segment lengths) to enable parallel decoding of bit-stream segments corresponding to particular sub-bit-planes, bit-planes, sub-bands, and decomposition levels, as the relevant coefficients are requested by the transform.

The storage and manipulation of the numerous independent bit-stream segments that are generated and accessed in parallel raise important complexity issues that are beyond the scope of this short paper. In particular, the complexity assessment in Section 7 does not include the bit-stream management. We note, however, that these issues are not unique to the present approach, as all scalable low-memory image compression schemes require some intermediate bit-stream storage, and must, to varying degrees, perform similar off-line bit-stream management.

In this particular architecture, the most challenging component of the bit-stream management sub-system is the interface between the coding engine and the bit-stream storage. This interface must be designed so that newly generated/requested portions of bit-stream segments are off-loaded/accessed in an efficient manner. The post-compression processing of bit-stream segments consists largely of memory transfers.

This architecture is related to the one considered in [2], with the key difference being that [2] is centered around a non-scalable coder which does not allow for the possibility of off-line reordering of the bit-stream, as the emphasis is on minimizing the overall latency of a combined encoder-decoder image communication system. Here, we consider applications in which encoding and decoding are decoupled.

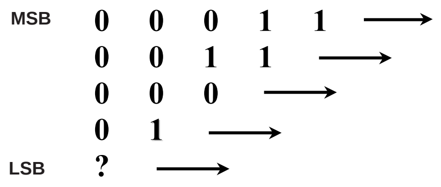

Figure 2: Stair-casing effect arising at decoder when non-scan-causal contexts are used for sub-bit-plane classification and coding.

3 Sequential bit-plane coding and scan-causality

Among reported image compression algorithms, those based on context-dependent coding of bit-planes (e.g., LZC [3] and CREW [4]) are most compatible with the above line-based architecture. We shall refer to such coders as *sequential bit-plane coders*. The coding engine for the proposed scheme can be classified as a sequential bit-plane coder with some important distinguishing properties that are motivated by practical aspects of the line-based architecture, and which are discussed next.

One property that is a fundamental requirement of the line-based architecture is the use of independently adapting probability models for compressing each sub-bit-plane and bit-plane of each sub-band and decomposition level. In contrast, most previously reported sequential bit-plane coders, which largely ignore the memory issues central here, compress the bit-planes sequentially, and carry probability model parameters across bit-plane boundaries.

Another property of the proposed coder, namely scan-causal modeling, addresses the following practical consideration. If the previously encoded bits of *all* spatially neighboring coefficients were used for sub-bit-plane classification and conditioning context for the current bit, the decoder would first need to decode the relevant bits of some "future" coefficients in order to decode the current bit. A stair-casing effect arises in the bit-plane dimension as the look-ahead decoding propagates to the neighbors and the neighbors' neighbors and so on. Figure 2 illustrates this phenomenon for a one-dimensional signal in which the more significant bits of scan-future nearest neighbors are part of the conditioning contexts. The figure illustrates the decoder look-ahead that must be performed for decoding the least significant bit "?" of the first coefficient. We see that, in this case, the extent of the stair-casing is equal to the number of bit-planes. Parallel or pipelined decoding of the bit-planes further increases the effective stair-casing extent to twice the number of bit-planes.

In the line-based setting, stair-casing in the vertical direction is very costly in terms of memory size and bandwidth, since the intermediate stair-case results need to be stored and accessed multiple times. Stair-casing in the horizontal direction is a nuisance, but it can be absorbed into the coding engine with a resulting increase in on-chip memory requirements. Horizontal stair-casing does make it more difficult to modify the coding scan to interface with a transform implementation that generates more than one line of coefficients at a time, which, as noted above, may be desirable from a memory bandwidth point of view.

The proposed coder, unlike previously reported sequential bit-plane coders, avoids the troublesome stair-casing effect, and the accompanying processing and memory bandwidth requirements, by using scan-causal, as opposed to information-causal contexts for sub-bit-plane classification and modeling. Scan-causal modeling amounts to requiring that the context used for the sub-bit-plane classification and coding of a given bit be based only on more significant bits of the current coefficient and/or equally or more significant bits of previously processed coefficients. Each coefficient is *scanned only once*, and all context information is derived from previously scanned coefficients. More significant bits from scan-future neighboring coefficients, even though they will appear earlier in the final embedded bit-stream, are not used in compressing the current bit. This modification clearly leads to sub-optimal compression efficiency since information ultimately available as conditioning context is ignored.

Summary. We can now form a clear picture of the tradeoffs involved in the proposed line-based coding algorithm. Among the significant advantages of the line-based architecture and scan-causal modeling are: 1) the inherent parallelism, which allows for low-latency processing and the potential to encode and decode one transform coefficient per clock cycle, and 2) a reduction of memory and bandwidth costs to the point where they are dominated by the transform, which allows for additional complexity tradeoffs by way of the transform filter lengths. These advantages, however, are not without cost as scan-causal modeling and independent model adaptation should incur some performance penalties relative to unconstrained approaches. Somewhat surprisingly, as we shall see in Section 6, this cost is minimal.

Comparison with other approaches. We conclude this section by comparing the proposed line-based coder architecture to other recently proposed classes of scalable image compression algorithms. Block-based coders, or those that encode disjoint blocks of transform coefficients [7] (using, e.g., quad-trees), are also good candidates for low-memory applications. Nevertheless, the external memory cost associated with buffering up lines into blocks is substantial when compared with the cost of line-based coding, as advocated here. Perhaps an even more important issue is that of parallelism. A key ingredient for achieving parallel encoding or decoding is that the evolution of the probability models within each processing unit proceed independently. Although the temptation for block coders is to scan the block multiple times[2] (once for each sub-bit-plane) in order to maximize compression efficiency, block coders do possess an obvious opportunity for parallelism in that each block is encoded and/or decoded independently. However, this "macroscopic" parallelism requires duplication of the block working memory in each processing engine, which can rapidly become expensive. Line-based coders conforming to the above architecture, however, exhibit "microscopic" parallelism. In this case, each sub-bit-plane is processed with a separate coding engine, with

[2]This is feasible for block coders, because blocks are understood as being small enough to fit locally on-chip.

the relevant models being adapted independently. This microscopic parallelism can generally be realized with substantially less internal memory than that associated with parallel block coding engines.

Finally, zero-tree-based coders such as EZW [5] and SPIHT [6], as originally formulated, require buffering up the entire image in order to generate an embedded bit-stream. While modifications of the basic algorithms can lead to lower memory implementations (with a possible negative effect on compression performance), the joint compression of "trees" of transform coefficients requires the synchronization of the transform across all decomposition levels, which, in turn, requires substantial buffering of intermediate transform data, even for a nominal number of decomposition levels. Therefore, zero-tree-based schemes are largely incompatible with the low-memory goals of this paper. Zero-tree-based schemes also do not directly yield bit-streams that are simultaneously resolution and quality scalable, a key functionality we wish to provide.

4 Modeling details

In this section we provide the details of the scan-causal context modeling used in the proposed coder. A major practical consideration in arriving at the proposed context models is that they be extremely simple, since, as noted above, separate models are maintained and updated in parallel for each sub-bit-plane.

We assume that the reader is familiar with the basic elements of bit-plane coding, such as the concepts of significance and refinement decisions [5, 3], and the idea of sub-bit-planes [1]. Adhering to the scan-causal modeling approach described above and assuming that each band is scanned from top to bottom and left to right, the conditioning contexts for modeling and for sub-bit-plane classification of the bits of the current coefficient X are based only on the neighboring coefficients (w, nw, n, ne), respectively corresponding to the coefficients directly left, left and above, above, and above and right of X. Let N denote this neighborhood of four coefficients. Three sub-bit-planes are formed within each bit-plane. Each significance decision is classified as predicted-significant or predicted-insignificant and the refinement decisions form a third sub-bit-plane. Let $\sigma_p(x)$ be a binary valued function, where $\sigma_p(x) = 1$ if coefficient x is significant with respect to a bit-plane greater than p, or becomes significant in bit-plane p and the significance decision is classified into the predicted-significant sub-bit-plane of p. A significance decision for coefficient X in bit-plane p is recursively classified into the predicted-significant sub-bit-plane of p if $\sigma_p(x) > 0$ for any neighboring coefficients $x \in N$. The remaining significance decisions in bit-plane p are classified into the predicted-insignificant sub-bit-plane.

Empirically validated theoretical considerations [1, 8] imply that the predicted-significant sub-bit-plane results in the greatest decrease in distortion per rate of description, followed by the predicted-insignificant sub-bit-plane, and finally by the refinement sub-bit-plane, with the latter two being very nearly inter-changeable in most of the bit-planes. Therefore, in order to generate an optimal SNR scalable bit-stream, the off-line bit-stream reordering step should place the bit-stream segment describing the predicted-significant sub-bit-plane before the predicted-insignificant bit-stream segment, which, in turn, should be placed before the refinement bit-stream segment.

The context for coding a bit in the predicted-significant sub-bit-plane of bit-plane p is derived from the evaluation of $\sigma_p(x)$ on $x \in N$. It follows from the definition of the predicted-significant sub-bit-plane that $\sigma_p(\cdot)$ is non-zero on at least one coefficient in N. Within each sub-band we obtain 3 contexts as follows.

LH band	HL band	HH band	Context
$\sigma_p(w) = 0$	$\sigma_p(n) = 0$	$\sigma_p(nw) = \sigma_p(ne) = 0$	1
$\sigma_p(w) = 1$	$\sigma_p(n) = 1$	$\sigma_p(nw) + \sigma_p(ne) \geq 1$	2
$\sigma_p(x) = 1$ for all $x \in N$	$\sigma_p(x) = 1$ for all $x \in N$	$\sigma_p(nw) + \sigma_p(ne) = 2$	3

The contexts for the predicted-insignificant sub-bit-plane are defined in terms of the binary function $\sigma'_p(x)$, where $\sigma'_p(x) = 1$ if coefficient x just becomes significant in bit-plane p. The non-zero contexts ($\sigma'_p(x) \neq 0$ for some $x \in N$) for the predicted-insignificant sub-bit-plane of bit-plane p are obtained by replacing $\sigma_p(\cdot)$ with $\sigma'_p(\cdot)$ in the above table. Intuitively, the all-zero context ($\sigma'_p(x) = 0$ for all $x \in N$) seems to provide less information about the current significance decision. It is therefore best to avoid the assumption of conditional independence given adjacent neighbors and to seek out additional conditioning information from non-adjacent neighbors. To this end, we partition this context into two contexts, based on whether the number of coefficients that have been processed since the last non-zero context in bit-plane p is larger than a fixed threshold (8 in our implementation).

Signs are encoded immediately after the corresponding coefficients are found to be significant. Sacrificing a small amount of coding efficiency for reduced complexity, we opted to code signs using a uniform $(1/2, 1/2)$ probability assignment. We also opted to pack the refinement sub-bit-plane uncoded into the relevant bit-stream segments.

5 Low-complexity coding

The final element of our low-complexity line-based package is the binary coder used to compress the significance sub-bit-planes. Like the context model, this coder should be extremely simple, since, in a hardware implementation, the inherent parallelism of the line-based approach necessitates having several such coders on a chip. Possible candidates are Golomb coding and the Q-coder arithmetic coding variants [10]. Golomb coding was already applied in [1] to coding runs of 0's in the sub-bit-planes, which were modeled as memoryless sources. The slightly more complex context models used here, however, render Golomb coding less appropriate since it is not well suited for coding binary sequences with context-dependent symbol probabilities.

An especially attractive candidate is the skew-coder [9], which is the ancestor of the Q-coder, QM-coder, and other arithmetic coding variants. The skew-coder's simplicity derives from certain approximations to the usual arithmetic coding operations. Very briefly, probability interval widths corresponding to strings ending in LPSs (least probable symbols) are restricted to be powers of two, and hence the updating of the A and C registers (the probability and code word registers as used in the arithmetic coding literature) upon encoding and decoding a binary symbol is reduced to adding or subtracting 2^{L-k}, where A and C are $L+1$ and L bits wide. The parameter k is called the skew-parameter and is typically determined adaptively from the statistics of the binary source, roughly so that 2^{-k} approximates the LPS probability.

It was observed in [9] that the skew-coder with fixed skew-parameter k reduces to Golomb coding with parameter $m = 2^k - 1$ when used in a single context. This suggests that with a properly chosen adaptation strategy, it should be possible to implement the skew-coder so that it operates at the complexity level of a Golomb coder when coding MPSs (most probable symbols) in a run of consecutive occurrences of a single context. This can in fact be done by ensuring that the adaptation of k coincides with register renormalization, and by making use of a simple function of the coder registers which indicates the number of MPSs that can be processed before renormalization is necessary. In our application, the all-zero predicted-insignificant context, which tends to occur in long contiguous runs, especially within the more significant bit-planes, is an ideal target for such an *accelerated skew-coder*. We note that the acceleration property of the skew-coder is useful primarily in software since it boosts average performance, while hardware must be designed for the worst case.

A similar acceleration technique has been suggested in connection with the Q and QM arithmetic coders ("speedup-mode" in [10]). The skew-coder, however, retains a unique simplicity concerning this "speedup-mode" in that the required division operation [10] can be implemented by masking an appropriate set of bits in the A and C registers. This simplification relative to the Q and QM coders stems from the power-of-two restriction mentioned above.

The above two properties of the skew-coder, namely the simple underlying operations and the accelerated mode, permit very efficient implementations of the line-based coding algorithm in both hardware and software. For this reason we adopted the skew-coder as the binary coder.

Adaptation. The skew parameter k can be adapted using a variety of simple count-based methods or state-machines. We opted to use the MELCODE state-machine from the run-mode of the JPEG-LS standard [11].

6 Compression efficiency

We evaluated the compression efficiency of two variants of the proposed coder on a set of four images (luminance only) used in the JPEG 2000 standardization process. The results were obtained using a 16-bit fixed point implementation of the (9,7) bi-orthogonal wavelet transform with 5 decomposition levels. Each sub-band was scaled so that the corresponding reconstruction basis functions in the image domain would have unit norm. The above line-based compression algorithm was then applied to the resulting coefficients one line at a time ($K = 1$ in the architecture description of Section 2). Finally, for each image, the reconstructions at each bit-rate were obtained by truncating a *single* sufficiently high-rate bit-stream.

The compression results are presented in Table 1. In addition to results for the proposed algorithm we have included results for three benchmark algorithms: sequential bit-plane coder A (a variation of the proposed algorithm), sequential bit-plane coder B, and SPIHT with arithmetic coding [6]. Sequential bit-plane coder A is a variation of the proposed coder that uses a full-fledged arithmetic coder to compress the binary decisions, and signs and refinement decisions are compressed according to simple context models derived from the scan-causal neighboring coefficients. Sequential bit-plane coder B is a full-fledged sequential bit-plane coder that has all of the elements of coder A and adds horizontally and vertically non-scan-causal neighbors to the coding and sub-bit-plane classification context-modeling. The context modeling of coder B also incorporates some non-nearest neighbors. Sequential bit-plane coder B is very similar to the entropy coder described in [12], except here it is applied to the bit-planes of *scalar* quantized indices. Coder A was in fact obtained by simply eliminating the non-nearest neighbors and non-scan-causal neighbors from the coding and sub-bit-plane classification context modeling of coder B.

From Table 1 we get a rough idea of the cost, in terms of compression efficiency, of the simplifications behind the proposed algorithm. At .5 bits per symbol, for example, we see that dropping non-causal horizontal and non-causal vertical neighbors results in a loss of about .15 db typically, and dropping the modeling of signs and refinement decisions and using the skew-coder instead of an arithmetic coder results in another .15 db loss typically.

The last line of Table 1 gives the average short-fall in PSNR of the proposed algorithm relative to the best average benchmark at each rate. The resulting short-falls correspond to about a 5% rate redundancy across the board.

We include a comparison with SPIHT only to convey an understanding of the compression efficiency of the proposed coder relative to a well-known benchmark. As noted above, however, zero-tree-based schemes such as SPIHT are not compatible with a low-memory architecture. In fact, even the performance of coder B is limited by complexity constraints that are not shared by SPIHT, such as the line-based *scanning order*. We also note that the published version of SPIHT used to obtain the numbers in Table 1 requires buffering up the whole image for compression and decompression.

7 Complexity analysis

Our analysis is based on the (9,7) transform that was used to obtain the above compression results. We assume the ability to process K lines of transform coefficients simultaneously with our line-based coder, although the compression performance was determined only for

Image		.0625	.125	.25	.5	1.0	2.0
aerial2	$*$	24.43	26.40	28.42	30.49	33.13	37.92
	†	24.49	26.43	28.53	30.57	33.36	38.36
	‡	24.53	26.46	28.57	30.61	33.45	38.43
	●	24.63	26.52	28.49	30.60	33.32	38.22
cafe	$*$	18.86	20.59	22.77	26.29	31.31	38.41
	†	18.85	20.62	22.82	26.45	31.57	38.84
	‡	18.90	20.66	22.94	26.58	31.84	39.22
	●	18.95	20.67	23.03	26.49	31.74	38.91
bike	$*$	23.14	25.63	28.95	32.80	37.35	43.42
	†	23.07	25.47	28.95	32.92	37.50	43.63
	‡	23.14	25.62	29.15	33.15	37.81	43.81
	●	23.44	25.89	29.12	33.01	37.70	43.80
woman	$*$	25.28	27.08	29.53	33.04	37.76	43.63
	†	25.32	27.08	29.50	33.12	37.84	43.65
	‡	25.25	27.14	29.67	33.28	38.11	43.75
	●	25.43	27.33	29.95	33.59	38.28	43.99
AVERAGE	$*$	22.93	24.92	27.42	30.65	34.89	40.84
	†	22.93	24.90	27.45	30.76	35.07	41.12
	‡	22.96	24.97	27.58	30.91	35.30	41.30
	●	23.11	25.10	27.65	30.92	35.26	41.23
Ave. Diff.		-.18	-.18	-.23	-.27	-.41	-.46

Table 1: PSNR's for the proposed coder and three benchmarks. ($*$)-proposed coder; (†)-sequential bit-plane coder benchmark A; (‡)-sequential bit-plane coder benchmark B; (●)-SPIHT with arithmetic coding.

$K = 1$. We are certain, however, that processing $K > 1$ lines at a time can only improve compression performance since the context model adaptation would benefit from a more spatially localized scan. In quoting memory size and memory bandwidth we assume that image pixels and transform coefficients are respectively 8 and 16 bits (a 16 bit fixed point transform was used to obtain the above results). The analysis also assumes a single image component (i.e. luminance only). Finally, we assume that all buffers that are image-size-dependent are contained in external memory.

External Memory. It can be shown [7] that the external memory required by the line-based transform for storing image pixels and intermediate samples from the Low-Low band of each decomposition level is equivalent to roughly $6K + 21$ image lines. The proposed coder needs to save an additional 5 bits of context information per column, from the most recent strip of K lines of transform coefficients in each band of each decomposition level. Under the above assumptions this amounts to an additional $(2)(3)(5/8)(1/2) \approx 1.9$ image lines. The total is about $(23 + 6K)$ lines.

External Memory Bandwidth. The external memory bandwidth for a straightforward implementation of the line-based wavelet transform is shown in [7] to be about $2.1 + 5.8/K$. As above, we add the extra bandwidth required for reading and writing the 5 context bits per column per strip of K lines for all bands and all levels to the external memory bandwidth of the line-based transform. The additional external memory bandwidth is thus $((2)(10/8)(3)(1/2))/(2K) = 1.9/K$ bytes per image sample. Another source of external memory bandwidth, unique to this coder, is the potential need for storage and retrieval to and from external memory of the coder state for each level in the wavelet decomposition. Let M be the number of bytes of coder state that need to be stored or retrieved per K lines of transform coefficients. If H and W are, respectively, the height and width of the original image, the total number of coder state external byte transfers is $(2)(M)(H/(2K) + H/(4K) + \ldots) = 2MH/K$, or $2M/(KW)$ transfers per pixel. The total then comes to $2.1+(7.7+(2M/W))/K$. This suggests that K should be no smaller than 2 or 4 and perhaps as large as 8 in order to approach the lower bound for external memory bandwidth, which is an extremely important parameter in most practical systems which work with large images. Smaller values of K might be reasonable when working with transform kernels whose vertical support is smaller than that of the (9,7) kernel.

Internal Memory. The bulk of the internal memory of the proposed line-based coding engine arises from the multiple coding units that must operate in parallel for each sub-bit-plane. For the two significance decision sub-bit-planes, the requirements amount to: 6 16-bit registers for the skew-coder variables; 8 16-bit symbol counters, one for each of the 8 contexts; 8 5-bit registers for model parameter variables; 8 1-bit flags for maintaining MPS/LPS to 0/1 mapping; and 2 32-bit i/o shift registers. The total is 336 bits. Since the refinement sub-bit-plane is left uncoded, it requires only an additional 32 bit i/o shift register, for a total of 368 bits for one bit-plane. Assuming that 10 bit-planes will be preserved for each coefficient, we conclude that 3680 bits of internal memory are required per band. Finally, multiplying this number by 3, we get 11040 bits of internal memory for the coding engine. We note however that there is a great deal of flexibility in the design of this internal unit. For example, reducing the symbol counters from 16 bits to, say, 10 or even 8 bits for the non-zero contexts (all but one), would probably have no impact on the overall compression efficiency. This would also permit the reduction of two of the skew-coder registers to 10 bits as well. This set of numbers leads to slightly over $(316)(10)(3) = 9480$ bits of internal memory to handle the three sets of K lines of transform coefficients in each of the three high-pass bands.

Internal memory bandwidth and throughput. An ideal implementation of the proposed coding engine is as a very wide circuit that simultaneously processes all bit-planes of the current coefficient and updates all of the relevant state variables in a single clock cycle. A small amount of pipe-lining may be necessary to achieve this throughput guarantee.

Acknowledgments. This paper emerged as a result of the authors' participation in the on-going JPEG2000 image compression standardization process. Wei Yu is acknowledged for a skew-coder implementation and discussions on its acceleration.

8 References

[1] E. Ordentlich, M. J. Weinberger, and G. Seroussi, "A low-complexity modeling approach for embedded coding of wavelet coefficients," in *Proc. 1998 IEEE Data Compression Conference*, (Snowbird, Utah, USA), pp. 408–417, Mar. 1998.

[2] C. Chrysafis and A. Ortega, "Line Based, Reduced Memory, Wavelet Image Compression," in *Proc. 1998 IEEE Data Compression Conference*, (Snowbird, Utah, USA), pp. 398–407, Mar. 1998.

[3] D. Taubman and A. Zakhor, "Multirate 3-D subband coding of video," *IEEE Transactions on Image Processing*, 3(5), pp. 572–588, Sept. 1994.

[4] A. Zandi, J. D. Allen, E. L. Schwartz, and M. Boliek, "CREW: Compression with reversible embedded wavelets," in *Proc. 1995 IEEE Data Compression Conference*, (Snowbird, Utah, USA), pp. 212–221, Mar. 1995.

[5] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Transactions on Signal Processing*, 41(12), pp. 3445–3462, Dec. 1993.

[6] 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*, pp. 243–250, 6(3), June 1996.

[7] D. Taubman. Embedded Block Coding with Optimized Truncation. ISO/IEC JTC 1/SC 29/WG1 document N 1020R, October 21, 1998.

[8] J. Li and S. Lei, "Rate-distortion optimized embedding," *Picture Coding Symposium*, Berlin, Germany, pp. 201–206, Sep. 10-12, 1997.

[9] G. G. Langdon Jr. and J. Rissanen, "A simple general binary source code," *IEEE Transactions on Information Theory*, 28(5), pp. 800–803, Sept. 1982.

[10] W. B. Pennebaker and J. L. Mitchell. *JPEG Still Image Data Compression Standard.* Van Nostrand Reinhold, New York, New York, 1993.

[11] ISO/IEC 14495-1, ITU T.87, "Information technology - Lossless and near-lossless compression of continuous-tone still images," 1998. Final Draft International Standard.

[12] P. Sementilli, A. Bilgin, J. H. Kasner, and M. W. Marcellin, "Wavelet TCQ: Submission to JPEG 2000," (invited paper), *Proceedings, Applications of Digital Image Processing*, SPIE, San Diego, California, July 1998.