Embedded Block Coding in JPEG2000

David Taubman
The University of New South Wales, Sydney, Australia

Erik Ordentlich
GlobeSpan, Santa Clara, California

Marcelo Weinberger and Gadiel Seroussi
Hewlett-Packard Laboratories, Palo Alto, California

January 16, 2001

Abstract

This paper describes the embedded block coding algorithm at the heart of the JPEG2000 image compression standard. The paper discusses key considerations which led to the development and adoption of this algorithm, and also investigates performance and complexity issues. The JPEG2000 coding system achieves excellent compression performance, somewhat higher (and, in some cases, substantially higher) than that of SPIHT with arithmetic coding, a popular benchmark for comparison. The algorithm utilizes the same low complexity binary arithmetic coding engine as JBIG2. Together with careful design of the bit-plane coding primitives, this enables comparable execution speed to that observed with the simpler variant of SPIHT without arithmetic coding. The coder offers additional advantages including memory locality, spatial random access and ease of geometric manipulation.

1 Introduction

JPEG2000 [1] is a new image compression standard, developed under the auspices of ISO/IEC JTC1/SC29/WG1 (commonly known as the JPEG committee). The standard departs radically from its better known predecessor, JPEG [2]. In place of the DCT (Discrete Cosine Transform), JPEG2000 employs a DWT (Discrete Wavelet Transform). Whereas arithmetic coding and successive approximation are options in JPEG, they are central concepts in JPEG2000. The coding mechanisms themselves are more efficient and support more flexible, finely embedded representations of the image. The JPEG2000 algorithm also inherently supports good lossless compression, competitive compression of bi-level and low bit-depth imagery, and bit-streams which embed good lossy representations of the image within a lossless representation.

JPEG2000 places a strong emphasis on scalability, to the extent that virtually all JPEG2000 bit-streams are highly scalable. In order to support the needs of a wide variety of applications, different progression orders are defined. The scalability property, in its different forms, pertains to the *ordering* of information within the bit-stream. However, as discussed next, the *coding* process plays a key role. In general, dependencies introduced during this process can destroy one or more degrees of scalability. Thus, while the DWT provides a natural framework for scalable image compression, the coding methods described in this paper are key to the realization of the potential derived from this framework. Therefore, one goal of this Introduction is to precisely define the main notions of scalability involved, discussing their implication in the design of the coding scheme.

A resolution-scalable bit-stream is one from which a reduced resolution may be obtained simply by discarding unwanted portions of the compressed data. The lower resolution representation should be identical to that which would have been obtained if the lower resolution image were compressed directly. The DWT is an important tool in the construction of resolution-scalable bit-streams. As shown in Figure 1, a first DWT stage decomposes the image into four subbands, denoted LL_1, HL_1 (horizontally high-pass), LH_1 (vertically high-pass) and HH_1. The next DWT stage decomposes this LL_1 subband into four more subbands, denoted LL_2, LH_2, HL_2 and HH_2. The process continues for some number of stages, D, producing a total of $3D + 1$ subbands whose samples represent the original image. The total number of samples in all subbands is identical to that in the original image.

The DWT's multi-resolution properties arise from the fact that the LL_d subband is a reasonable low resolution rendition of LL_{d-1}, with half the width and height. Here, the original image is interpreted as an LL_0 subband of highest resolution, while the lowest resolution is represented directly by the LL_D subband. The LL_d subband, $0 \leq d < D$, may be recovered from the subbands at levels $d + 1$ through D by applying only $D - d$ stages of DWT synthesis. So long as each subband from DWT stage d, $0 < d \leq D$, is compressed without reference to information in any of the subbands from DWT stages d', $0 \leq d' < d$, we may convert a compressed image into a lower resolution compressed image, simply by

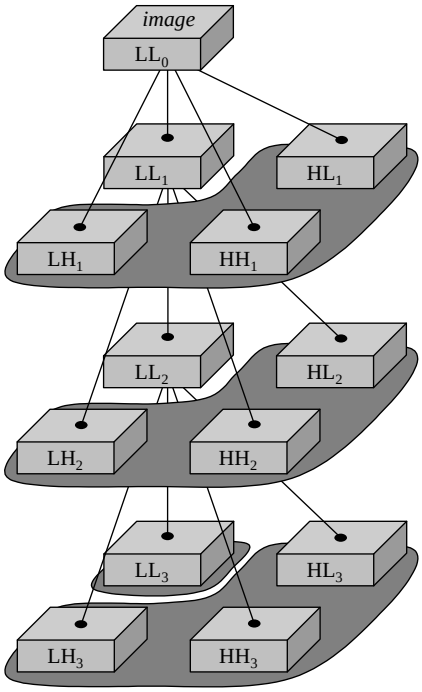

Figure 1: DWT with $D = 3$ stages.

discarding those subbands which are not required. The number of resolutions available in this way is $D + 1$.

A second type of scalability arises when the compressed bit-stream contains elements which can be discarded in order to obtain a lower quality (higher distortion) representation of the subband samples. We refer to this as distortion scalability. Ideally, the reduced quality representations obtained by discarding appropriate elements from a distortion scalable bit-stream can be decoded to reconstruct the original image with a fidelity approaching that of an "optimal" coder, tailored to produce the same bit-rate as the scaled bit-stream. Most practical means of achieving this goal involve some form of bit-plane coding, whereby the magnitude bits of the subband samples are coded one by one from most significant to least significant. Discarding least significant bits is equivalent to coarser quantization of the original subband samples. The terms "SNR scalability", "successive approximation" and "bit-rate scalability" have also been used in connection with this type of scalability.

Although resolution scalability (the ability to discard high frequency subbands) provides a crude mechanism for decreasing the bit-rate and increasing distortion, this is not usually an efficient mechanism for trading distortion for compressed size. It has been observed that discarding subbands from a compressed bit-stream generally produces lower resolution images with such small distortion (and large bit-rate) as to be inappropriate for applications requiring significant compression. In order to produce a family of successively lower image resolutions with a consistent level of perceived or objective distortion (e.g., a consistent mean squared error), the multi-resolution transform should be combined with distortion scalable coding.

Unfortunately, due to possible dependencies introduced during the coding process, the combination of a wavelet transform with bit-plane coding does not guarantee bit-streams that are both resolution-scalable and distortion-scalable. Furthermore, the order in which information appears within the compressed bit-stream can have a substantial impact on the resources required to compress or decompress a large image. The *zero-tree* coding structure [3] provides us with a useful example of the adverse consequences of excessive interaction between coding and ordering. Shapiro's original EZW algorithm [3] and Said and Pearlman's significantly enhanced SPIHT algorithm [4] provide excellent examples of embedded image compression. These algorithms have rightly received tremendous attention in the image compression community. However, the coding dependencies introduced by these algorithms dictate a distortion-progressive ordering of the compressed bits, as zero-trees involve downward dependencies between the subbands produced by successive DWT stages. These dependencies interfere with resolution scalability: no subset of the embedded bit-stream corresponds to the result of compressing a lower resolution image. Moreover, the encoder and decoder typically require a random access buffer, with storage for every subband sample in the image. Once compressed in this manner, the bit-stream cannot be reordered so as to support decompressors with reduced memory resources.

The JPEG standard also involves coding dependencies which prohibit some

useful orderings. In its hierarchical refinement mode, multi-resolution image hierarchies are represented using a Laplacian pyramid structure which requires lower resolutions to be fully decoded before meaningful decoding of a higher resolution image can take place. This representation interferes with the distortion scalability offered by JPEG's successive approximation mode, since it is not possible to decompress a subset of the bit-planes across all resolution levels. This problem is dual to the one observed for zero-tree coding. In JPEG's progressive modes, any scalable bit-stream necessarily involves multiple scans through the entire image. Moreover, these progressive scans use different coding techniques to those specified by the sequential mode. As a result, they cannot generally be collapsed back into a sequential representation without transcoding the compressed bit-stream.

The arguments advanced above suggest that one should endeavour to decouple the process of efficiently coding subband samples from the ordering of the compressed bit-stream. The separation of information coding and information ordering is indeed a key consideration in the design of the JPEG2000 algorithm. As a result, and in contrast to the above examples, the JPEG2000 standard supports spatially progressive organizations which allow decompressors to work through the image from top to bottom. Information may progress in order of increasing resolution, in order of increasing quality across all resolutions, or in sequential fashion across all resolutions and qualities. The progression order is independent of the coding techniques and may be adjusted at will, without recourse to transcoding. JPEG2000 also allows resource constrained decompressors to recover a reduced resolution version of an image which may be too large to decompress in its entirety.

Of course, it is not possible to completely decouple the coding and ordering of information, since efficient coding necessarily introduces dependencies. Sources of such dependencies include the use of conditional coding contexts, indivisible codes (e.g., vector, run-length or quad-tree codes) and adaptive probability models. There is also a limit to the granularity at which we can afford to label individual elements of the compressed bit-stream for subsequent reordering.

A natural compromise is to partition the subband samples into small blocks and to code each block independently. The various dependencies described above may exist within a block but not between different blocks. The size of the blocks determines the degree to which one is prepared to sacrifice coding efficiency in exchange for flexibility in the ordering of information within the final compressed bit-stream. This block coding paradigm is adopted by JPEG2000, based on the concept of *Embedded Block Coding with Optimal Truncation* (EBCOT) [5]. Each block generates independent bit-streams, which are packed into *quality layers*. In order to generate the quality layers, the independent bit-streams are in turn subdivided into a large number of "chunks." While preserving the ordering of chunks within a block, the compressor is free to interleave chunks from the various blocks in any desired fashion, thus assigning incremental contributions from each block to each quality layer. The independent bit-streams can be truncated at the end-points of these chunks, which are referred to as truncation points.

The selection of truncation points raises, again, an ordering problem, since it affects the rate-distortion properties of the overall image representation. In a bit-plane coding scheme, bit-plane end-points are natural truncation points for the embedded bit-stream. However, the availability of a finer embedding, with many more useful truncation points, is a key element in the success of the EBCOT paradigm. To achieve a finer embedding, the sequence in which bits from different samples are coded is data dependent. This sequence tends to encode the most valuable information (in the sense of reducing the distortion of the reconstructed image the most) as early as possible. The embedded block coder uses context modeling to address both the ordering and the arithmetic coding of the events. The concept of adaptive ordering through context modeling was introduced independently and in somewhat different forms in [6] and [7]. It is also closely related to the coding sequence employed in the SPIHT [4] and, to a lesser extend, EZW [3] algorithms. Rather than pursuing a totally adaptive approach, as in [7], JPEG2000 imposes reasonable assumptions on the data, defining context-dependent "fractional bit-planes", in the spirit of [6].

Thus, the block coding concept in JPEG2000 and the embedded coder itself draw heavily from the EBCOT algorithm [5], which itself builds upon the contributions of other works; however, there are some notable differences as well as a number of mode variations which can have significant practical implications. In this paper, our goal is to provide the reader with an appreciation for the salient features of the algorithm, as well as some of the considerations which have contributed to its development.

The rest of this paper is organized as follows. In Section 2, we discuss the EBCOT paradigm and its advantages. In Section 3, we present the primitive coding operations which form the foundation of the embedded block coding strategy. In Section 4, we introduce the concept of fractional bit-planes, and discuss the principles behind it. In Section 5, we provide some indication of the performance of the algorithm, while in Section 6 we discuss its complexity, both for software and hardware implementations. Finally, in Section 7, we present variations on the algorithm, which are supported by Part 1 of the standard.

2 The EBCOT Paradigm

2.1 Independent Code-Blocks

Within the EBCOT paradigm adopted by JPEG2000, each subband is partitioned into relatively small blocks (e.g., 64×64 or 32×32 samples) which we call "code-blocks." This is illustrated in Figure 2. Each code-block, $\mathcal{B}_i$, is coded independently, producing an elementary embedded bit-stream, $\mathbf{c}_i$. It is convenient to restrict our attention to a finite number of allowable truncation points, $Z_i + 1$, for code-block $\mathcal{B}_i$, having lengths, $L_i^{(z)}$, with

$$0 = L_i^{(0)} \leq L_i^{(1)} \leq \cdots \leq L_i^{(Z_i)} .$$

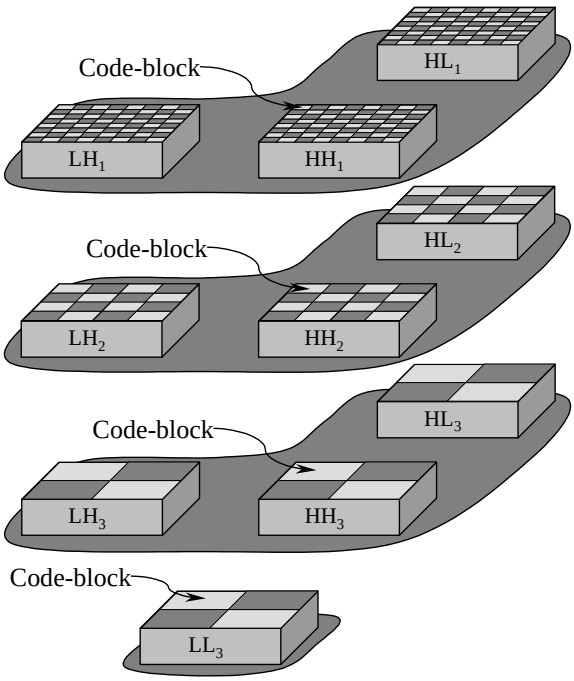

Figure 2: Division of subbands into code-blocks. Here, code-blocks have the same dimensions in every subband.

In the present development we are not concerned with the details of the embedded block coding algorithm, or the determination of these truncation points; these are the subject of Sections 3 and Section 4.

We assume that the overall reconstructed image distortion can be represented as a sum of distortion contributions from each of the code-blocks and let $D_i^{(z)}$ denote the distortion contributed by block $\mathcal{B}_i$, if its elementary embedded bit-stream is truncated to length $L_i^{(z)}$. Calculation or estimation of $D_i^{(z)}$ depends upon the subband to which block $\mathcal{B}_i$ belongs. For most of the ensuing discussion, however, we may simply consider the image as being composed of a collection of blocks, $\mathcal{B}_i$, without regard for the subbands to which their samples belong.

Since the code-blocks are compressed independently, we are free to use any desired policy for truncating their embedded bit-streams. If the overall length of the final compressed bit-stream is constrained by $L_{\max}$, we are free to select any set of truncation points, $\{z_i\}$, such that

$$\sum_i L_i^{(z_i)} \leq L_{\max}$$

Of course, the most attractive choice is that which minimizes the overall distor-

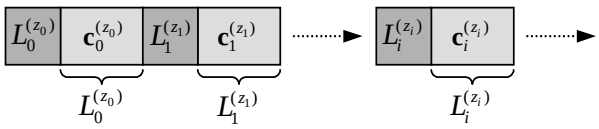

Figure 3: Simple pack-stream formed by concatenating optimally truncated code-block bit-streams.

tion,

$$D = \sum_i D_i^{(z_i)}$$

The selection of truncation points may be deferred until after all of the code-blocks have been compressed, at which point the available truncation lengths, $L_i^{(z)}$, and the associated distortions, $D_i^{(z)}$, can all be known. For this reason, we refer to the optimal truncation strategy as one of post-compression rate-distortion optimization (PCRD-opt). For details of the PCRD-opt algorithm, the reader is referred to [5].

A chief disadvantage of independent block coding would appear to be that it is unable to exploit redundancy between different blocks within a subband or between different subbands. In fact, an important premise of zero-tree algorithms such as EZW and SPIHT is that substantial redundancy exists between "parent" and "child" samples within the subband hierarchy. Somewhat surprisingly, these disadvantages are more than compensated by the fact that the contributions of each code-block to the final bit-stream may be independently optimized by the PCRD-opt algorithm.

2.2 Quality Layers

The overall compressed bit-stream is constructed by packing contributions from the various code-block bit-streams, $\mathbf{c}_i$, together in some fashion. We use the term "pack-stream," to distinguish this overall representation from the individual block bit-streams. The simplest pack-stream organization consistent with the EBCOT paradigm is illustrated in Figure 3. In this case, the optimally truncated block bit-streams, $\mathbf{c}_i^{(z_i)}$, are simply concatenated, with length tags inserted to identify the contribution from each code-block.

This simple pack-stream is resolution-scalable, since each resolution level consists of a well-defined collection of code-blocks, each of which is explicitly identified by means of the length tags. The pack-stream also possesses a degree of spatial scalability. So long as the subband synthesis filters have finite support, each code-block influences only a finite region in the reconstructed image. Thus, given a spatial region of interest, the relevant code-blocks may be identified and extracted from the pack-stream.

The simple pack-stream of Figure 3 is not distortion-scalable, even though its individual code-blocks have embedded representations. The problem is that the

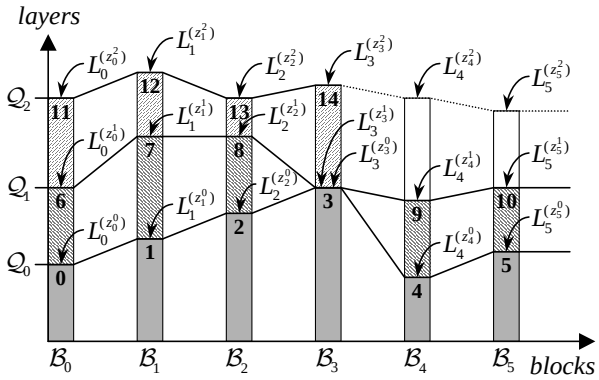

Figure 4: Quality layers in JPEG2000. Numbers indicate the sequence of code-block contributions required for a quality progressive pack-stream.

pack-stream offers no information to assist in the construction of a smaller pack-stream whose code-block contributions are optimized in any way. To resolve this difficulty, the EBCOT algorithm [5] introduces a quality layer abstraction, as illustrated in Figure 4. Only 6 code-blocks are shown for the sake of illustration. There are a total of Λ quality layers, labeled $\mathcal{Q}_0$ through $\mathcal{Q}_{\Lambda-1}$. The first layer, $\mathcal{Q}_0$, contains optimized code-block contributions, having lengths $L_i^{(z_i^0)}$, which minimize the distortion, $D^0 = \sum_i D_i^{(z_i^0)}$, subject to a length constraint, $\sum_i L_i^{(z_i^0)} \leq L_{\max}^0$. Subsequent layers, $\mathcal{Q}_\lambda$, contain additional contributions from each code-block, having lengths $L_i^{(z_i^\lambda)} - L_i^{(z_i^{\lambda-1})}$, which minimize the distortion,

$$D^\lambda = \sum_i D_i^{\left(z_i^\lambda\right)}$$

subject to a length constraint,

$$\sum_i L_i^{\left(z_i^\lambda\right)} \leq L_{\max}^\lambda \,.$$

Although each quality layer conceptually contains a contribution from every code-block, we emphasize the fact that some or even all of these contributions may be empty. In the example of Figure 4, code-block $\mathcal{B}_3$ makes no contribution to layer $\mathcal{Q}_1$. A distortion-scalable pack-stream may be constructed by including sufficient information to identify the contribution made by each code-block to each quality layer. Moreover, quality progressive organizations are clearly supported by sequencing the information in the manner suggested by the numbering in Figure 4.

If a quality progressive pack-stream is truncated at an arbitrary point, the decoder can expect to receive some number of complete quality layers, followed

by some fraction of the blocks from the next layer. In the example of Figure 4, the third quality layer, $\mathcal{Q}_2$, is truncated before code-block $\mathcal{B}_4$. In this case, the received prefix will not be strictly optimal in the PCRD-opt sense. However, this form of sub-optimality may be rendered negligible by employing a large number of layers. On the other hand, more layers implies a larger overhead to identify the contributions made by each block to each layer.

When a large number of layers are used, some effort must be invested in efficiently coding the auxiliary information which identifies the various code-block contributions. JPEG2000 provides a "second tier" coding strategy for this type of information. The idea of separating the coding process into two tiers was introduced in [5] and indeed the second tier coding mechanisms used by JPEG2000 are essentially those described there. We shall not discuss them further in the present text.

2.3 EBCOT Advantages

At this point, it is worth summarizing some of the benefits which the EBCOT paradigm imparts to JPEG2000.

Flexible organization: EBCOT pack-streams possess resolution scalability, distortion scalability (so long as multiple quality layers are used) and a degree of spatial scalability. When multiple image components are compressed (e.g., colour components), these components form a fourth dimension of scalability. Progressions along all four dimensions are supported by the JPEG2000 standard.

Custom quality interpretations: Since each quality layer may contain arbitrary contributions from each of the code-blocks, the notion of quality may be adapted to application specific measures of significance. By contrast with EZW, SPIHT and other embedded compression algorithms, the EBCOT paradigm allows code-blocks to be marginalized or entirely suppressed in lower quality layers when the corresponding spatial regions or frequency bands are known to be less significant for some application.

Local processing: Independent coding allows local processing of the samples in each code-block, which is especially advantageous for hardware implementations. Independent coding also introduces the possibility of highly parallel implementations, where multiple code-blocks are encoded or decoded simultaneously. For very large images, spatially oriented progressions of the pack-stream may be used in conjunction with incremental processing of the subband/wavelet transform to facilitate "streaming". In this case, it is sufficient to buffer only a local window into the pack-stream, the image and its subbands. In this way, implementation memory can be much smaller than the image which is being compressed or decompressed. This same property allows for efficient rotation and flipping of the image during decompression.

Efficient compression: As noted above, the use of PCRD optimization can more than compensate for the small efficiency losses arising from the imposition of independent block coding. The algorithm is also able to accommodate spatially varying and/or image dependent measures of distortion. One interesting

example arises in visual perception, where local activity can mask the visibility of certain types of compression artifacts. A masking-sensitive distortion measure and promising experimental results are provided in [5].

Compressed domain manipulation: Cropping an image from any boundary affects only those subband samples whose synthesis waveforms intersect with the cropped image region. Code-blocks containing these samples must be re-coded, but the remaining (interior) code-blocks are unaffected by cropping. This, together with the DWT realignment capabilities offered by JPEG2000, allows images to be repeatedly cropped from any boundary without the build-up of compression artefacts commonly experienced with other schemes such as JPEG. It is also possible to flip, transpose and rotate images by multiples of 90°, by performing only local block transcoding operations. Repeated application of these transformations can also be free from compression noise build-up. For further details see [8].

Error resilience: Errors encountered in any code-block's bit-stream will clearly have no influence on the other blocks. This, together with the natural prioritization of information induced by embedded block coding and quality layers, allows for the construction of powerful unequal protection strategies for error prone environments.

3 Bit-Plane Coding

The coding of code-blocks in JPEG2000 proceeds by bit-planes. Bit-plane coding naturally arises in the framework of embedded quantization, as discussed in Section 3.1 below. In Section 3.2, we show how coding proceeds in order to derive an embedded bit-stream, and we discuss the importance of data dependent ordering strategies for achieving a fine embedding of the information. The remainder of the section is devoted to a detailed study of the primitive context modeling and coding operations which form the foundation of the embedded block coding strategy.

3.1 Embedded Quantization

Since each code-block is to be represented by an efficient embedded bit-stream, prefixes of the bit-stream must correspond to successively finer quantization of the block's sample values. In fact, the underlying quantizers are inevitably embedded [9, §4B]. In the ensuing discussion we restrict our attention to the practically appealing case of embedded deadzone quantization.

A deadzone quantizer with step size Δ yields quantization indices

$$q = \mathrm{sign}(x) \left\lfloor \frac{|x|}{\Delta} + \tau \right\rfloor \tag{1}$$

where x denotes a subband sample from the code-block and τ is a parameter controlling the width of the central deadzone. When $\tau = \frac{1}{2}$, the quantizer is

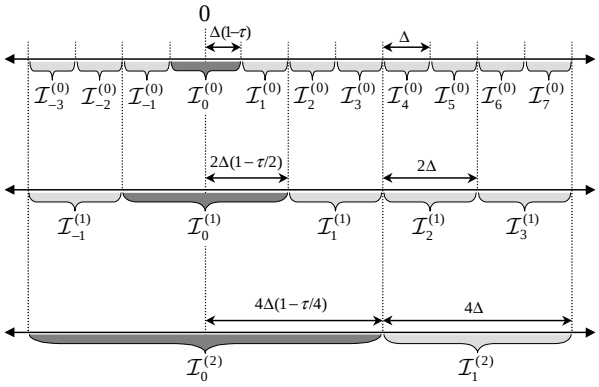

Figure 5: Family of embedded deadzone scalar quantizers.

uniform, while $\tau = 0$ corresponds to the case in which the deadzone width is 2Δ. The quantization intervals, denoted $\mathcal{I}_q^{(0)}$, are illustrated in Figure 5.

Let $\chi = \text{sign}\,(x)$ and $v = |q|$ denote the sign and magnitude of q.[1] Also, let

$$v^{(p)} = \left\lfloor \frac{v}{2^p} \right\rfloor$$

denote the value formed by dropping p LSB's (Least Significant Bits) from $v = v^{(0)}$. Employing the easily verified identity,

$$\left\lfloor \frac{\lfloor a \rfloor}{b} \right\rfloor = \left\lfloor \frac{a}{b} \right\rfloor, \quad \forall a \in \mathbb{R} \text{ and } b \in \mathbb{N}$$

we see that χ and $v^{(p)}$ are the sign and magnitude of the index, $q^{(p)}$, obtained using the coarser quantizer

$$q^{(p)} = \text{sign}\,(x) \left\lfloor \frac{|x|}{2^p \Delta} + \frac{\tau}{2^p} \right\rfloor$$

Figure 5 illustrates the corresponding quantization intervals, $\mathcal{I}_{q^{(p)}}^{(p)}$.

This family of deadzone quantizers has three notable characteristics: 1) the step sizes are given by $\Delta^{(p)} = 2^p \Delta$; 2) the deadzone width parameters, $\tau^{(p)} = 2^{-p}\tau$, rapidly converge to 0 as p increases; and 3) each quantization interval, $\mathcal{I}_{q^{(p)}}^{(p)}$, is embedded within a coarser interval, $\mathcal{I}_{q^{(p+1)}}^{(p+1)}$. In view of property (2), it makes sense to restrict our attention to the case $\tau = 0$. In this case, all quantizers have the same structure, with a deadzone twice as wide as the other intervals. It is worth noting, however, that the coding techniques described in

[1] Strictly speaking, when $q = 0$ the sign of q is indeterminate; this will be reflected in the fact that it is not coded. It is convenient here to associate χ with the sign of the original subband sample, y, which is the same as that of q whenever $q \neq 0$.

this chapter are applicable to the more general case in which $\tau \neq 0$. In fact, Part 2 of the JPEG2000 standard is expected to support such general deadzone quantizers.

3.2 Coding and Ordering

Let $x[\mathbf{j}] \equiv x[j_1, j_2]$ denote the sequence of subband samples belonging to the relevant code-block, having height J_1 and width J_2, so that $0 \leq j_1 < J_1$ and $0 \leq j_2 < J_2$. Similarly, let $\chi[\mathbf{j}]$ and $v^{(p)}[\mathbf{j}]$ denote the sign and magnitude of the corresponding embedded quantization indices. Suppose that K is a sufficient number of bits to represent any of the quantization index magnitudes, meaning that $v^{(K)}[\mathbf{j}] = 0$ for all $\mathbf{j}$. Finally, let $v^p[\mathbf{j}] \in \{0, 1\}$ be the LSB of $v^{(p)}[\mathbf{j}]$, which is also bit p of $v[\mathbf{j}]$. We say that bits $v^p[\mathbf{j}]$ from all samples in the code-block constitute "magnitude bit-plane" p. There are at most K non-trivial magnitude bit-planes. The value of K is signalled separately for every code-block, when that code-block first contributes to the JPEG2000 pack-stream. In fact, K itself is coded in a manner which exploits redundancy between adjacent code-blocks within the same subband. The particular coding technique is known as "tag tree coding". For further details regarding such second tier coding of code-block summary information, the reader is referred to [5] and [1].

An embedded bit-stream may be formed in the following way. First, code the most significant magnitude bit-plane, $v^{K-1}[\mathbf{j}]$, together with the sign, $\chi[\mathbf{j}]$, of any sample for which $v^{K-1}[\mathbf{j}] \neq 0$. If the bit-stream is truncated at this point, the decoder can reconstruct the coarsest quantization indices, $q^{(K-1)}[\mathbf{j}]$. Then code the next most significant magnitude bit-plane, $v^{K-2}[\mathbf{j}]$, including the sign of any sample for which $v^{K-2}[\mathbf{j}] = 1$ and $v^{(K-1)}[\mathbf{j}] = 0$. Proceed in this way for each magnitude bit-plane, p, including the sign of those samples for which $v^p[\mathbf{j}]$ is the most significant non-zero bit. We refer to this process as bit-plane coding and we use the term "bit-plane" loosely to refer to both the magnitude and associated sign information. If the bit-stream is truncated at the end of bit-plane p, the decoder can reconstruct quantization indices, $q^{(p)}[\mathbf{j}]$.

A variety of techniques may be employed to code the magnitude and sign bits. An efficient bit-plane coder, however, should exploit the substantial redundancy which generally exists between successive bit-planes. This goal may be achieved using conditional arithmetic coding, which requires the definition of a scanning order and a context model. Early bit-plane coders [10, 11] processed the quantized subband samples following a deterministic scan (line by line) within each bit-plane[2]. In principle, the order in which the information is coded should have no impact on coding efficiency, since the code length assigned by any conditional probability model (through arithmetic coding) can be matched by another model which uses an arbitrary scanning order, by appropriate decomposition of the corresponding joint distribution. However, such a decomposition may involve statistical dependencies between each coded symbol

[2] Although not originally described as such, zero-tree algorithms such as SPIHT and EZW also amount to bit-plane coding algorithms. The statement here concerning early bit-plane coders is not intended to include such algorithms.

and all previously coded symbols in the current and previous bit-planes. Thus, in practice, our adaptive assignment of conditional probabilities may result in code lengths that do depend on the sequence of coding events. Nevertheless, we find empirically that the particular probability models used by JPEG2000, which are described in Section 3.3, yield code lengths for each bit-plane which are largely insensitive to the order in which information is coded.

In [12], bit-plane coding is formalized as a sequence of steps aimed at providing the next coded sample (for efficient embedding), and a corresponding conditional probability distribution (for efficient coding). The empirical observation above suggests that, as proposed in [6], efficient embedding (rather than efficient coding) should be the decisive consideration in selecting the scanning order. In the case of a deterministic scan, bit-plane end-points are the only natural truncation points for the embedded bit-stream. Truncation at any other point must yield an expected distortion-rate pair which lies strictly above the convex distortion-rate curve associated with deadzone scalar quantization. As explained in Section 2.2, quality layers are constructed by applying a PCRD-opt algorithm to optimize the code-block truncation points. In order to provide a larger number of useful truncation points, thereby enhancing the effectiveness the PCRD-opt algorithm, a finer embedding is required than that offered by deterministically scanned bit-plane coders. To achieve such an embedding, information is coded in a data dependent order. This order tends to encode the most valuable information (in the sense of reducing the distortion of the reconstructed image the most) as early as possible [7, 6, 13, 5].

Following [6] and [7], the embedded block coder adopted by JPEG2000 uses context modeling to address both the ordering and coding of information within each code-block. Moreover, JPEG2000 imposes reasonable assumptions on the data, defining context-dependent "fractional bit-planes," in the spirit of [6]. The specific determination of fractional bit-planes is described in Section 4. The reader should note that the adaptive ordering of information within each code-block is based on information available to both the encoder and decoder so that it need not be signalled explicitly. Of course, this also means that the compressor has no control over the coding order. This is quite different to the ordering of code-block contributions within the pack-stream, as described in Section 2.2.

3.3 Conditional Arithmetic Coding of Bit-Planes

In this section we describe the bit-plane coding primitives defined by the JPEG2000 image compression standard. At any given sample location, $\mathbf{j}$, in any given bit-plane, p, we must code the value of $v^p[\mathbf{j}]$ and possibly also the sign, $\chi[\mathbf{j}]$. These are binary events and we employ an adaptive binary arithmetic coder. The specific arithmetic coding variant employed by JPEG2000 is the MQ coder, which is discussed further in Section 6.1. For our present discussion it is sufficient to understand the arithmetic coder as a "machine", which efficiently represents a sequence of binary outcomes subject to the provision of good probability estimates. The adaptive probability models evolve within a number of distinct

contexts, which depend upon information which has already been coded. The specification of these models is identical to that proposed in [5], being independent of the order in which information is actually coded[3]. Ordering considerations are deferred until Section 4.

Image subband samples tend to exhibit distributions which are heavily skewed toward small amplitudes. As a result, when $v^{(p+1)}[\mathbf{j}] = 0$, meaning that $x[\mathbf{j}] \in \mathcal{I}_0^{(p+1)}$, we can expect that $x[\mathbf{j}]$ is also very likely to be found in the smaller deadzone, $\mathcal{I}_0^{(p)}$. Equivalently, the conditional PMF, $f_{V^p|V^{(p+1)}}(v^p|0)$, is heavily skewed toward the outcome $v^p = 0$. For this reason, an important element in the construction of efficient coding contexts is the so-called "significance" of a sample, defined by

$$\sigma^{(p)}[\mathbf{j}] = \begin{cases} 1 & \text{if } v^{(p)}[\mathbf{j}] > 0 \\ 0 & \text{if } v^{(p)}[\mathbf{j}] = 0\,. \end{cases}$$

To decouple our description of the coding operations from the order in which they are applied, we introduce the notion of a binary "significance state," $\sigma[\mathbf{j}]$. At any point in the coding process, $\sigma[\mathbf{j}]$ assumes the value of $\sigma^{(p)}[\mathbf{j}]$ where p is the most recent (least significant) bit for which information concerning sample $x[\mathbf{j}]$ has been coded. Equivalently, we initialize the significance state of all samples in the code-block to 0 at the beginning of the coding process and then toggle the state to $\sigma[\mathbf{j}] = 1$ immediately after coding the first non-zero magnitude bit for sample $x[\mathbf{j}]$.

We identify three different types of primitive coding operations as follows. If $\sigma[\mathbf{j}] = 0$ we refer to the task of coding $v^p[\mathbf{j}]$ as "significance coding," since $v^p[\mathbf{j}] = 1$ if and only if the significance state transitions to $\sigma[\mathbf{j}] = 1$ in this coding step. In the event that the sample does become significant, we must invoke a "sign coding" primitive to identify $\chi[\mathbf{j}]$. For samples which are already significant, the value of $v^p[\mathbf{j}]$ serves to refine the decoder's knowledge of the non-zero sample magnitude. Accordingly, we invoke a "magnitude refinement coding" primitive.

3.3.1 Significance Coding (Normal Mode)

The significance coding primitive involves a normal mode and a run mode. We describe the normal mode first. In this mode, one of 9 different contexts is used to code the significance (i.e., the value of $v^p[\mathbf{j}]$) of a sample which is currently insignificant (i.e., $v^{(p+1)}[\mathbf{j}] = 0$). Context selection is based upon the significance of the sample's 8 immediate neighbours.

The context label, $\kappa^{\mathrm{sig}}[\mathbf{j}]$, is formed from three intermediate quantities,

$$\begin{aligned} \kappa^{\mathrm{h}}[\mathbf{j}] &= \sigma[j_1, j_2 - 1] + \sigma[j_1, j_2 + 1] \\ \kappa^{\mathrm{v}}[\mathbf{j}] &= \sigma[j_1 - 1, j_2] + \sigma[j_1 + 1, j_2] \\ \kappa^{\mathrm{d}}[\mathbf{j}] &= \sum_{k_1=\pm 1} \sum_{k_2=\pm 1} \sigma[j_1 + k_1, j_2 + k_2]\,. \end{aligned}$$

[3] The one exception to this rule is given by the run mode specified in Section 3.3.2.

Samples which lie beyond the boundaries of the relevant code-block are regarded as insignificant for the purpose of constructing these three quantities. Evidently, there are 45 possible combinations of the three quantities, $\kappa^{\mathrm{h}}[\mathbf{j}]$, $\kappa^{\mathrm{v}}[\mathbf{j}]$ and $\kappa^{\mathrm{d}}[\mathbf{j}]$. A context reduction function is used to map these 45 combinations into 9 distinct context labels, $\kappa^{\mathrm{sig}}[\mathbf{j}]$. Details of the context reduction mapping may be found in [5] or [1]. It suffices here to note that the mapping is sensitive to the orientation of the subband to which the relevant code-block belongs.

3.3.2 Significance Coding (Run Mode)

At moderate to high compression ratios, most of the subband samples must be insignificant in all of the bit-planes which are actually included in the final packstream. To see this, observe that whenever a sample becomes significant we must code the significance event (usually with respect to a conditional PMF skewed heavily toward insignificance) and also the sign. The combined cost of these two binary events is unlikely to be less than 2 bits and may be considerably more. Even those samples which are eventually coded as significant, may be insignificant for many of the initial bit-planes.

Since code-block samples are expected to be predominantly insignificant, a run mode is introduced to dispatch multiple insignificant samples with a single binary symbol. The run mode serves primarily to reduce complexity, although very minor improvements in compression performance are also typical. The run mode is entered if the probability of significance is determined to be sufficiently small. This determination is based on the stripe-based scan discussed in Section 4 and depicted in Figure 6. Specifically, the run mode is entered if and only if the following three conditions hold simultaneously.

1. Four consecutive samples (following the scan shown in Figure 6) must currently be insignificant. That is, $\sigma[\mathbf{j}_r] = 0$ for $0 \le r < 4$, where $\mathbf{j}_0 = \mathbf{j}$ and $\mathbf{j}_r$ is the r^{th} position beyond $\mathbf{j}$ in the scan.

2. All four samples must currently have insignificant neighbourhoods. That is, $\kappa^{\mathrm{h}}[\mathbf{j}_r] + \kappa^{\mathrm{v}}[\mathbf{j}_r] + \kappa^{\mathrm{d}}[\mathbf{j}_r] = 0$ for $0 \le r < 4$.

3. The group of four samples must be aligned on a four sample boundary within the scan. As we shall see in Section 4, the scanning pattern itself works column by column on stripes of four rows at a time. This means that the samples must constitute a single stripe column.

In run mode, a binary "run interruption" symbol is coded to indicate whether or not all four samples remain insignificant in the current bit-plane, p. Insignificance is identified by the symbol 0, while a value of 1 means that at least one of the four samples becomes significant. The run interruption symbol is coded within its own context, denoted $\kappa^{\mathrm{run}} = 9$.

If one or more of the four samples becomes significant during the current bit-plane, p, the insignificant run length, r, must also be coded, followed by the sign of the first significant sample, $\chi[\mathbf{j}_r]$. The remaining samples are then coded in

normal mode, until the conditions required for run mode are encountered again. Experience shows that the run length is nearly uniformly distributed, which is also to be expected if samples transition to significance with very low probability. For this reason the 2-bit run-length, r, is coded one bit at a time, starting with the most significant bit, using a non-adaptive uniform probability model.

3.3.3 Sign Coding

The sign coding primitive is invoked at most once for any sample, $x[\mathbf{j}]$, immediately after the significance coding operation in which the sample first becomes significant. Most algorithms proposed for coding subband sample values, whether embedded or otherwise, treat the sign as an independent, uniformly distributed random variable, devoting 1 bit to coding its outcome. It turns out, however, that the signs of neighbouring sample values exhibit significant statistical redundancy. Some arguments to suggest that this should be the case are presented in [14] and [5].

The JPEG2000 sign coding primitive employs 5 contexts. Context design is based upon the relevant sample's immediate four neighbours, each of which may be in one of three states: significant and positive; significant and negative; or insignificant. There are thus 81 unique neighbourhood configurations. For details of the symmetry conditions and approximations used to map these 81 configurations to one of 5 context labels, $\kappa^{\mathrm{sign}}[\mathbf{j}]$, the reader is referred to [5].

3.3.4 Magnitude Refinement Coding

The magnitude refinement primitive is used to code the next magnitude bit, $v^p[\mathbf{j}]$, of a sample which is already significant; i.e., $\sigma^{(p+1)}[\mathbf{j}] = 1$. This information refines the coarser quantization index, $q^{(p+1)}[\mathbf{j}]$, to the next finer index, $q^{(p)}[\mathbf{j}]$. As already noted, subband samples tend to exhibit symmetric distributions, $f_X(x)$, which are heavily skewed toward $x = 0$. In fact, the conditional PMF $f_{V^p|Q^{(p+1)}}(v^p \mid q^{(p+1)})$ typically exhibits the following characteristics: 1) it is independent of the sign of $q^{(p+1)}$; 2) $f_{V^p|Q^{(p+1)}}(0 \mid q^{(p+1)}) > \frac{1}{2}$ for all $q^{(p+1)}$; and 3) $f_{V^p|Q^{(p+1)}}(0 \mid q^{(p+1)}) \approx \frac{1}{2}$ for large $\left|q^{(p+1)}\right|$.

As a result, it is desirable to condition the coding of $v^p[\mathbf{j}]$ upon the value of $v^{(p+1)}[\mathbf{j}]$ when $v^{(p+1)}[\mathbf{j}]$ is small. We also find that it can be useful to exploit redundancy between adjacent sample magnitudes when $v^{(p+1)}[\mathbf{j}]$ is small. These observations serve to justify the assignment of one of 3 coding contexts, κ^{mag}, as follows.

$$\kappa^{\mathrm{mag}}[\mathbf{j}] = \begin{cases} 0 & \text{if} \quad v^{(p+1)}[\mathbf{j}] = 1 \\ & \text{and} \quad \kappa^{\mathrm{h}}[\mathbf{j}] + \kappa^{\mathrm{v}}[\mathbf{j}] + \kappa^{\mathrm{d}}[\mathbf{j}] = 0 \\ 1 & \text{if} \quad v^{(p+1)}[\mathbf{j}] = 1 \\ & \text{and} \quad \kappa^{\mathrm{h}}[\mathbf{j}] + \kappa^{\mathrm{v}}[\mathbf{j}] + \kappa^{\mathrm{d}}[\mathbf{j}] > 0 \\ 2 & \text{if} \quad v^{(p+1)}[\mathbf{j}] > 1 \end{cases} \tag{2}$$

4 Fractional Bit-Plane Scan

In this section, we specify the order in which samples are visited when a given bit-plane is scanned. As discussed in Section 3.2, this order is data dependent, and is aimed at improving the embedding of the code-stream. This goal is achieved through multiple coding passes. For each bit-plane, p, the coding proceeds in a number of distinct passes, which we identify as "fractional bit-planes": $\mathcal{P}_1^p$, $\mathcal{P}_2^p$ and $\mathcal{P}_3^p$. Each coding pass involves a scan through the code-block samples in stripes of height 4, as shown in Figure 6. This scan has been chosen to facilitate efficient software and hardware implementations of the standard [15]. Some of the advantages of a stripe-based scan will become apparent in Section 6 (the reader is referred to [15] for a more detailed discussion). Information for bit-plane p is coded for each sample in only one of the passes; that sample is skipped in the other two passes. Fractional bit-planes are treated as indivisible units, and thus determine the candidate truncation points for the code-stream. Membership of each of the three coding passes is determined dynamically, based upon the significance state of each sample's eight immediate neighbours. These are the same neighbours which are used to determine the conditional coding contexts described in Section 3.3.

4.1 Significance Propagation Pass

The first coding pass in each bit-plane, $\mathcal{P}_1^p$, includes any sample location, $\mathbf{j}$, which is itself insignificant, but has a significant neighbourhood; that is, at least one of its eight neighbours is significant. Membership in $\mathcal{P}_1^p$ may be expressed by the conditions $\sigma[\mathbf{j}] = 0$ and $\kappa^{\mathrm{h}}[\mathbf{j}] + \kappa^{\mathrm{v}}[\mathbf{j}] + \kappa^{\mathrm{d}}[\mathbf{j}] > 0$. These conditions are designed to include those samples which are most likely to become significant in bit-plane p. Moreover, for a broad class of probability models, including those typically used in image compression, the samples in this coding pass are likely to yield the largest decrease in distortion relative to the increase in code length [6, 12].

Each sample in the pass is coded using the significance coding primitive described in Section 3.3.1. The sign coding primitive is invoked immediately after any significance coding step in which the sample becomes significant; i.e., $v^p[\mathbf{j}] = 1$. It is worth noting that samples which become significant in this pass may give rise to waves of significance determination events which propagate along connected image features such as edges. This is because membership of the coding pass is assessed incrementally. Once a sample becomes significant, the four neighbours which have not yet been visited in the scan then also have significant neighbourhoods, and will be included in $\mathcal{P}_1^p$ unless they are already significant. We call this the "significance propagation pass" to remind the reader that its members are assessed dynamically.

Figure 6 provides an example of the significance propagation pass for one stripe of the code-block. In the figure, empty circles identify samples which are insignificant in bit-plane p; shaded circles identify samples which become significant during the pass, $\mathcal{P}_1^p$; and solid dots indicate samples which were

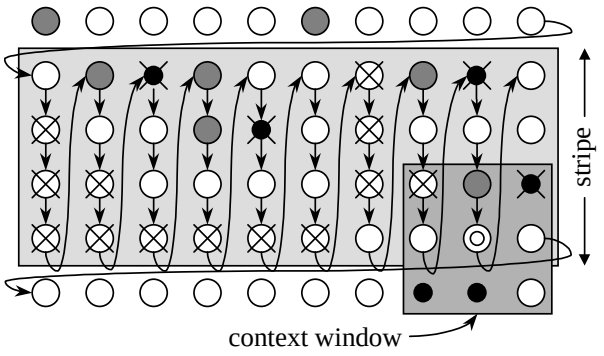

Figure 6: Stripe oriented scan. Refer to the discussion of the significance propagation pass for an explanation of the symbols used in this example.

already significant in bit-plane $p-1$. Crosses are used to mark those samples which do not belong to the significance propagation pass, either because they were already significant, or because their neighbourhood is entirely insignificant at the point in the scan when they are visited.

4.2 Magnitude Refinement Pass

In the second pass of each bit-plane, $\mathcal{P}_2^p$, the magnitude refinement primitive of Section 3.3.4 is used to code magnitude bit $v^p[\mathbf{j}]$ of any sample which was already significant in the previous bit-plane; i.e., $\sigma^{(p+1)}[\mathbf{j}] = 1$. Equivalently, $\mathcal{P}_2^p$ includes any sample whose significance state is $\sigma[\mathbf{j}] = 1$, which was not already included in $\mathcal{P}_1^p$.

4.3 Cleanup Pass

The final coding pass, $\mathcal{P}_3^p$, includes all samples for which information has not already been coded in bit-plane p. From the definitions of $\mathcal{P}_1^p$ and $\mathcal{P}_2^p$, we see that samples coded in this pass must be insignificant. The significance coding primitives described in Sections 3.3.1 and 3.3.2 are used to code $v^p[\mathbf{j}]$ for all samples belonging to this pass. We note that the conditions for run mode may occur only in this coding pass. As explained in Section 3.3.2, run mode is entered if an entire stripe column contains insignificant samples with entirely insignificant neighbours. The significance of all of these samples is coded in $\mathcal{P}_3^p$, using the run mode to identify the first if any of the samples which becomes significant in bit-plane p. Coding of any remaining samples in the stripe column proceeds in normal mode. As always, the sign coding primitive is invoked for any sample which becomes significant, immediately after its significance is coded.

4.4 Rate-Distortion Properties

As already mentioned, the available truncation points for the embedded bit-stream correspond to the coding pass end-points. Thus, the number of non-zero truncation points for code-block $\mathcal{B}_i$ is

$$Z_i = 3K_i - 2$$

where K_i is the number of magnitude bit-planes specified for code-block $\mathcal{B}_i$ (recall that the number of bit-planes is explicitly signalled for each code-block).

For each truncation point, $z \in \{1, 2, \ldots, Z_i\}$, the length, $L_i^{(z)}$, identifies the smallest prefix of the embedded bit-stream which is sufficient to correctly decode all symbols up to the end of coding pass $\mathcal{P}_k^p$, $0 \leq p < K_i$, where p, k and z are related through

$$z = 3\,(K_i - p) + k - 3\,.$$

The first available truncation point, $z = 0$, always corresponds to discarding the entire bit-stream so that $L_i^{(0)} = 0$.

As mentioned in Section 2, the rate-distortion properties of the overall compressed image representation, depend upon the selection of appropriate truncation points for each code-block. In particular, given any $s > 0$, any set of truncation points, $\{z_i\}$, which minimizes the functional,

$$\sum_i D_i^{(z_i)} + s \sum_i L_i^{(z_i)}$$

is optimal in the sense that it is not possible to further reduce the distortion without increasing the overall bit-rate. The value of s is selected so that the solution which minimizes this functional achieves the desired overall bit-rate or distortion.

Thus, for any given s, each z_i must minimize $D_i^{(z_i)} + sL_i^{(z_i)}$. Let $\mathcal{H}_i$ be the set of all truncation points for code-block $\mathcal{B}_i$, which are solutions to this optimization problem for some value of the parameter s. That is,

$$\mathcal{H}_i = \bigcup_{s>0} \left\{ z \mid D_i^{(z)} + sL_i^{(z)} \leq D_i^{(z')} + sL_i^{(z')}, \forall z' \right\}$$

As argued in [5] and explicitly shown in [8], $\mathcal{H}_i$ describes the vertices of the lower convex hull of the set of distortion-rate pairs, $\left(D_i^{(z)}, L_i^{(z)}\right)$. This is illustrated in Figure 7. Points in the interior of the convex hull will never be selected by an optimal assignment algorithm (i.e., a PCRD-opt algorithm).

One way to assess the suitability of a particular set of definitions for the fractional bit-plane coding passes, is to measure the frequency with which each of the truncation points, z, belongs to $\mathcal{H}_i$. At one extreme, we might find that $\mathcal{H}_i$ consists only of the bit-plane end-points (i.e., those truncation points, corresponding to the end of each cleanup pass, $\mathcal{P}_3^p$). This would mean that

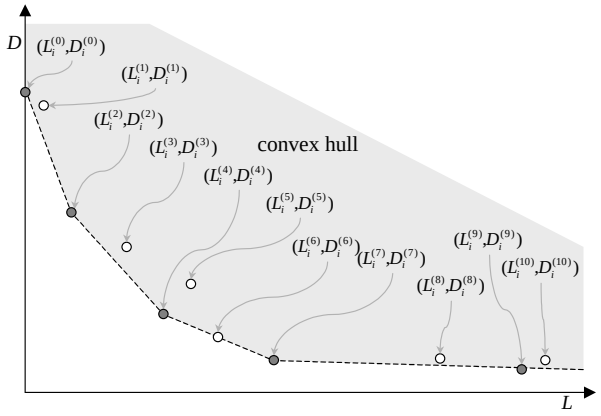

Figure 7: Convex hull of the distortion-rate pairs, $\left(D_i^{(z)}, L_i^{(z)}\right)$, for code-block $\mathcal{B}_i$. Solid dots identify the candidate points, $\mathcal{H}_i$, for optimal truncation.

de-interleaving the code-block samples into fractional bit-planes is useless. At the other extreme, we might find that every truncation point belongs to $\mathcal{H}_i$, meaning that every coding pass has a beneficial impact on the rate-distortion performance of the overall compression system.

In practice, neither of these extremes is observed. The convex hull occupancy (i.e., $\mathcal{H}_i$ occupancy) results shown in Figure 8 suggest that each of the three types of coding pass is frequently beneficial. These results are obtained by applying the JPEG2000 algorithm to the three large ISO/IEC photographic test images, "Bike", "Cafe" and "Woman", using a block size of 64×64. Experiments are run for various quantization step sizes, so as to cover the most interesting range of overall image bit-rates (measured in bits/sample). Notice that the bit-plane end-points (equivalently, the cleanup pass end-points) do indeed contribute most frequently to $\mathcal{H}_i$. The other two coding passes also contribute to the convex hull more often than not.

While the significance propagation pass should clearly precede the others, by virtue of its steeper distorion-length slope (change in distortion, divided by change in length), the situation is a priori less clear with respect to the order of the other two passes. The results in Figure 8, however, also provide justification for the fact that the magnitude refinement pass is best performed before the cleanup pass in JPEG2000. To see this, we observe that the distortion-length slope over coding pass $\mathcal{P}_2^p$ is most often steeper than that over $\mathcal{P}_3^p$. This is a necessary condition for the end-point of $\mathcal{P}_2^p$ to contribute to $\mathcal{H}_i$, which occurs much more often than not (see Figure 8). If the order of the magnitude refinement and cleanup passes were reversed, $\mathcal{P}_3^p$ would most often have a steeper distortion-length slope than $\mathcal{P}_2^p$, preventing the latter from contributing frequently to $\mathcal{H}_i$ and thereby weakening the embedding. This argument relies upon the assumption that the distortion-length slopes associated with magnitude refinement and

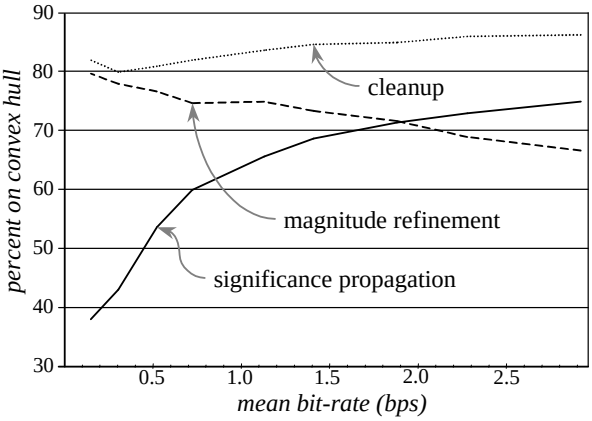

Figure 8: Convex hull occupancy rates for each of the three types of fractional bit-plane coding passes.

cleanup coding operations are not affected by their order, which is largely the case[4].

4.5 Other Variations

The idea of sequencing bit-plane coding steps in accordance with their anticipated distortion-length slopes was conceived independently by Li and Lei [7] and Ordentlich et al. [6]. It could be argued that the lists maintained by the SPIHT [4] algorithm serve a similar purpose, representing one of its most significant innovations over Shapiro's EZW algorithm [3]. Also, Li et al. [16] proposed a reordering of EZW's coding steps in accordance with their anticipated distortion-length slopes.

Li and Lei [7] proposed a more complex algorithm, in which the coding passes are not confined to bit-plane boundaries. The distortion-length slope is explicitly estimated for each neighbourhood context configuration, based on distortion models and probability estimates available from the adaptive arithmetic coder. Each coding pass incorporates those coefficient bits which have not previously been coded and whose distortion-length slope is estimated to be at least as steep as some threshold. The passes are thus implicitly defined by the sequence of thresholds. If the thresholds are close together, each coding pass represents coefficient bits which are expected to yield similar distortion-length slopes. In this way, it can happen that some particularly favourable samples are much

[4]The magnitude refinement coding context, κ^{mag}, does have some dependence upon the significance of the sample's neighbours and hence the order in which the refinement and cleanup passes are performed. It turns out, however, that this effect is usually quite small. In any event, the effect tends to strengthen the present argument, since delaying the magnitude refinement pass ensures that more information is available for coding, so that its distortion-rate slope can be even steeper.

more finely quantized than others at any given point in the embedding.

Ordentlich et al. [6] explored fractional bit-plane coding passes which eventually evolved into the scheme used in JPEG2000, in the context of a simple bit-plane coding scheme involving Golomb encoded run lengths. They defined coding passes whose membership is based on information available from previous bit-planes only. A fourth pass also included information from other subbands. The ideas in [6] were combined with conditional arithmetic coding of the bit-planes by Sheng et al. [13].

The above works were based on the coding of subbands as a whole. Independent code-blocks and incremental assessment of membership in the "significance propagation" pass were introduced by Taubman in [5]. That work also investigated other fractional bit-plane assignment rules. A four pass model incorporating a novel backward scanning pass was found to yield superior embedding to the three pass approach defined above for JPEG2000. Not only are there more truncation points, but these truncation points also contribute to the convex hull with greater frequency than that observed in Figure 8. In most cases, however, the four pass model was found to offer negligible improvement over the simpler three pass approach described above. A careful study of this and other refinements leading to the final form of the JPEG2000 algorithm may be found in [15].

5 Compression Performance

In this section we provide some indication of the performance of the JPEG2000 coder by comparing it with the SPIHT algorithm [4], which has become a popular benchmark for image compression. A 5 level DWT with the Cohen-Daubechies-Feauveau 9/7 biorthogonal wavelet kernels [17] is used for these experiments. Since both algorithms employ exactly the same wavelet transform and exactly the same quantization strategy, PSNR[5] results serve as a meaningful indication of coding efficiency.

The PSNR results reported in Table 1 are obtained using the JPEG2000 Verification Model (VM8.0) and the public domain implementations of SPIHT, both with and without arithmetic coding, which are available from "www.rpi.edu". The experiments are performed with distortion progressive representations of each image. In the JPEG2000 case, a single pack-stream is generated, having a quality progressive order. This single pack-stream is truncated to each of the indicated test bit-rates and decompressed. In addition to distortion scalability, the JPEG2000 pack-stream is also resolution-scalable and supports a degree of spatial random access. Moreover, it may be reordered to support spatially progressive organizations, for applications which cannot afford to keep the entire compressed representation in memory. The SPIHT bit-stream supports none of these additional features.

The first set of results reported in the table identifies average performance over the three most popular natural images from the JPEG2000 test set, "Bike",

[5] For 8-bit images, PSNR (measured in dB) is defined as $10 \log_{10}(255^2/\text{MSE})$.

Table 1: PSNR (dB) obtained by decompressing a single file, truncated to different bit-rates. SPIHT results are quoted relative to the PSNR observed with JPEG2000.

Category		.125 bps	.25 bps	.5 bps	1.0 bps
Natural	JPEG2000	24.83	27.58	31.32	36.19
(2560	SPIHT-AC	-0.20	0.21	-0.29	-0.27
× 2048)	SPIHT-NC	-0.67	-0.77	-0.94	-0.99
Chart	JPEG2000	28.80	32.50	37.48	43.74
(2347	SPIHT-AC	-1.15	-1.29	-1.23	-1.18
× 1688)	SPIHT-NC	-6.64	-4.82	-4.34	-3.73
Lenna	JPEG2000	31.04	34.14	37.30	40.40
(512	SPIHT-AC	+0.07	-0.00	-0.05	+0.06
× 512)	SPIHT-NC	-0.31	-0.42	-0.43	-0.37
Barbara	JPEG2000	25.43	28.40	32.22	37.16
(512	SPIHT-AC	-0.57	-0.82	-0.82	-0.74
× 512)	SPIHT-NC	-0.97	-1.18	-1.28	-1.22

"Cafe" and "Woman", each of size 2560×2048. The other results correspond to the 2347×1688 JPEG2000 test image, "Chart", and the more widely known 512×512 images, "Lenna" and "Barbara". We have selected this small set of test images primarily to emphasize key features of the coder. More comprehensive experimental results may be found in [18].

Evidently, the JPEG2000 coder exhibits somewhat higher compression efficiency, while offering substantially more flexibility than SPIHT. We draw the reader's attention to the fact that the compression performance of JPEG2000 is more robust to variations in image content than that of SPIHT (especially the version without arithmetic coding), as evidenced by the image "chart." This robustness follows from imposing fewer assumptions a priori on the structure of the wavelet coefficients being coded.

6 Complexity Considerations

Our purpose in this section is to suggest that the JPEG2000 coder is able to meet the demands of high performance applications. We begin with a brief discussion of the MQ arithmetic coder. Contrary to popular belief, arithmetic coding need not be a highly complex operation. We then provide evidence from our experience in working with software implementations of the standard. Finally, Sections 6.3 and 6.4 provide some useful statistics which may be used to predict throughput and buffering requirements for hardware implementations of the standard.

6.1 The MQ Coder

The binary-valued symbols produced by the bit-plane coding primitives discussed in Section 3.3 are coded using the MQ coder. The MQ coder is a

multiplier-free adaptive, binary arithmetic coder with renormalization-driven probability estimation. It includes the multiplier-free approximation and bit-stuffing policies introduced by the Q-coder [19], enhanced by conditional exchange and Bayesian learning in the probability estimation state machine[6].

At any given point in the coding process, the string of symbols which have been seen so far is mapped to a unique sub-interval, $[c, c+a) \subseteq [0,1)$, represented by

$$c = C \cdot 2^{-16-N} \text{ and } a = A \cdot 2^{-16-N} \tag{3}$$

where C and A are integers and N is the number of normalization shifts which have been employed to ensure that $2^{15} \leq A < 2^{16}$. Upon completion, the compressed bit-stream is the most significant $\approx N$ bits of C; encoder and decoder termination policies establish the actual number of MSB's which must be retained from C. This type of representation is common to virtually all realizations of the arithmetic coding principle.

Each coding context, κ, is represented by 7 bits of state information. One bit holds the identity, s_κ, of the MPS (Most Probable Symbol) for context κ. The remaining 6 bits identify the state, Σ_κ, of a machine, which estimates the LPS (Least Probable Symbol) probability for context κ; the machine has 47 different states. Since the JPEG2000 coder uses only 18 contexts, and each requires a 7-bit representation, a high performance hardware implementation may choose to maintain the context states in high speed registers.

Suppose the next symbol, x, occurs in context κ, for which the Least Probable Symbol (LPS) is currently estimated to occur with probability $p_{\Sigma_\kappa} \in (0, 1/2)$. Ideally, the interval length shrinks according to $a \leftarrow a p_{\Sigma_\kappa}$ if $x = 1 - s_\kappa$ (the LPS) and $a \leftarrow a - a p_{\Sigma_\kappa}$ if $x = s_\kappa$ (the MPS). These are approximated by $A \leftarrow \bar{p}_{\Sigma_\kappa}$ and $A \leftarrow A - \bar{p}_{\Sigma_\kappa}$, respectively, where $\bar{p}_{\Sigma_\kappa}$ is an integer approximation to $p_{\Sigma_\kappa} \alpha 2^{16}$ and $\alpha \approx 0.71$ is an empirically observed mean of $2^{-16}A$. The LPS is mapped to the lower sub-interval so that C is unchanged; when an MPS occurs, we map $C \leftarrow C + \bar{p}_{\Sigma_\kappa}$.

If this process leaves $A < 2^{15}$ a renormalization operation is applied to restore A to its legal range. During renormalization, both A and C are multiplied by 2 (i.e. left-shifted), incrementing N to preserve the validity of equation (3), until A is restored to the range $2^{15} \leq A < 2^{16}$. The state variables, s_κ and Σ_κ, are updated only when a symbol which is coded in context κ generates a renormalization event ($A < 2^{15}$).

Since renormalization adds at least one bit to the length of the arithmetic codeword (i.e., N) significant compression means that renormalization occurs rarely. Indeed, the majority of symbols are coded as MPS's and do not induce renormalization. For these symbols, the MQ coder performs only three simple steps: i) $A \leftarrow A - \bar{p}_{\Sigma_\kappa}$; ii) $C \leftarrow C + \bar{p}_{\Sigma_\kappa}$; and iii) a single bit test for the renormalization condition, $A < 2^{15}$. The more expensive operations occur only during

[6] These enhancements were introduced with the QM-coder, adopted by the JBIG and JPEG standards; probability estimation in the MQ-coder proceeds according to the state transition table known as JPEG-FA in [20].

renormalization; these include state transitions in the adaptive probability estimator, the renormalization shifts themselves, and the assembly of completed code-bytes.

In dedicated hardware implementations, it is even possible to achieve throughputs in excess of one symbol per clock cycle, by encoding or decoding consecutive symbols concurrently. This is possible so long as the concurrently coded symbols are not separated by a renormalization event. This possibility is described in [20, Chapter 13.7] as a "speedup mode" for the QM-coder defined by the JPEG standard. It is discussed further in [21]. Notice that the speedup mode has a similar goal as the run mode described in Section 3.3.2. However, while the run mode affects the bit-stream (and is thus mandatory), the speedup mode is a matter of implementation.

The normalization policy described above ensures that A may be represented using a 16-bit unsigned integer. At any point in the coding process, however, C has a $16 + N$ bit representation. Since C represents the lower bound of an interval whose length is represented by $A < 2^{16}$, the value C' represented by these same $16 + N$ bit positions at any subsequent point in the coding process must satisfy $C \leq C' < C + 2^{16}$. All arithmetic operations take place in the 16 LSB's of C, but we are prevented from immediately dispatching the more significant bits of C to an output buffer by the possibility that a carry bit generated by the arithmetic might propagate into these bits.

The QM-coder used by the JPEG and JBIG standards [20] stacks consecutive $\texttt{FF}_\text{h}$ bytes from the more significant bits of C indefinitely, until carries can be resolved. The MQ-coder, however, follows the Q-coder [19] in adopting a "bit-stuffing" approach, which allows code bytes to be dispatched in a more regular manner with lower implementation cost. Bytes are dispatched through a single byte buffer; whenever this buffer assumes the value $\texttt{FF}_\text{h}$, an extra bit is inserted into the representation of C so as to ensure that the effect of future coding steps may not propagate beyond the single byte stored in this buffer. Bit-stuffing adds approximately 0.05% to the code length, but has the desirable effect of bounding the number of operations which must be performed during renormalization.

The specific realization of bit-stuffing in the MQ-coder has the property that any consecutive pair of bytes dispatched to the compressed bit-stream is guaranteed to lie in the range $\texttt{0000}_\text{h}$ through $\texttt{FF8F}_\text{h}$. JPEG2000 exploits this property by defining unique marker codes in the range $\texttt{FF90}_\text{h}$ through $\texttt{FFFF}_\text{h}$, which may be used to enhance error resilience and facilitate parsing and reorganization of the code-stream. The reader is referred to [1] and [8] for further details regarding error resilience, the MQ coder and associated implementation techniques.

6.2 Software Experience

Table 2 provides timing figures for software implementations of the SPIHT and JPEG2000 algorithms. The conditions and implementations used to obtain these results are identical to those used to obtain the PSNR results in Table 1. The image category identified as "popular" refers to the popular test images,

Table 2: CPU time (μs/pel) obtained by decompressing a single file, truncated to different bit-rates, using a 400 MHz Pentium II processor.

Category		.125 bps	.25 bps	.5 bps	1.0 bps
Natural	JPEG2000	.041μs	.081μs	.157μs	.301μs
(2560	SPIHT-AC	.363μs	1.29μs	3.60μs	8.62μs
$\times$ 2048)	SPIHT-NC	.292μs	.928μs	2.84μs	8.63μs
Chart	JPEG2000	.053μs	.081μs	.153μs	.301μs
(2347	SPIHT-AC	.293μs	.974μs	2.84μs	5.94μs
$\times$ 1688)	SPIHT-NC	.033μs	.308μs	1.20μs	5.64μs
Popular	JPEG2000	.043μs	.067μs	.154μs	.290μs
(512	SPIHT-AC	.084μs	.114μs	.336μs	.728μs
$\times$ 512)	SPIHT-NC	.044μs	.054μs	.112μs	.338μs

Lenna and Barbara. Only the decoding process is considered (not including the DWT). This is due, in part, to the fact that the encoding time for JPEG2000 depends upon the policy used to determine the number of bit-planes which should actually be encoded for each code-block. Ideally, one would estimate this quantity with sufficient accuracy to avoid discarding most of the encoded data during PCRD optimization, when the optimal set of code-block truncation points is determined. Our motivation for not including DWT execution times in the results reported in Table 2 is that both SPIHT and JPEG2000 are employing exactly the same transform here. It is worth noting that the execution time required for block encoding or decoding tends to dominate that associated with a carefully optimized implementation of the DWT, except at very low bit-rates.

SPIHT has the advantage that the encoding process may be terminated as soon as the desired bit-rate has been achieved. The price paid for this, however, is that the encoding and decoding processes require non-local access to the image transform coefficients. This, in turn, causes a dramatic reduction in throughput as the image dimensions grow, since the machine spends most of its time performing non-local memory accesses. Table 2 clearly demonstrates this sensitivity to image size. Interestingly, the throughput of the JPEG2000 algorithm is competitive with the uncoded version of SPIHT, even when working with small images.

6.3 Hardware Implementation

The JPEG2000 block coder has been designed with hardware implementation in mind. As part of the process which led to the selection and refinement of this algorithm by the JPEG committee, some of the expected properties of such an implementation were investigated. Some discussion of these matters may be found in [22] and [15]. In this section, we outline one possible high level architecture for the block coder, along with measured statistics which may be used to predict some aspects of its performance.

Figure 9 provides a high level overview of a possible implementation of the

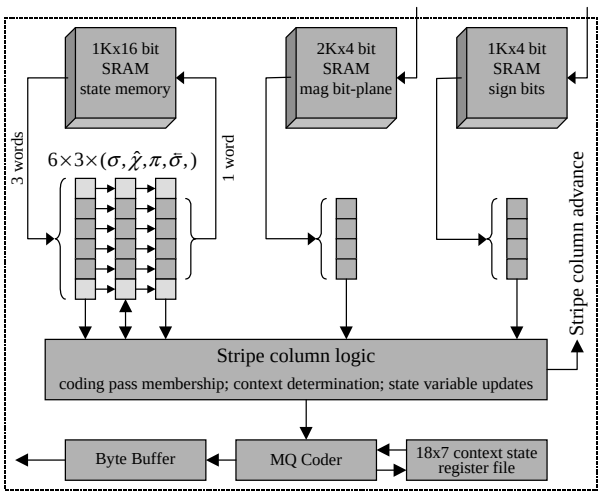

Figure 9: Bit-plane oriented block coding architecture. Decoder is similar with reversal of some data flows.

block coder, which exploits the regular stripe oriented coding scan shown in Figure 6. Interestingly, there is no need to buffer the samples of an entire code-block in internal memory. Instead, we shall assume here that the relevant quantized subband samples are available in an external memory and that they are already separated into bit-planes: a sign bit-plane and K magnitude bit-planes. As discussed in Section 6.4, such a memory organization leads naturally to a scheme for controlling the external memory bandwidth associated with intermediate buffering of subband samples. The block coder imports the sign bits and the most significant magnitude bit-plane into an internal memory. It then processes the most significant magnitude bits, while loading the next most significant bit-plane and so forth.

All internal memories are organized into words which represent a single stripe column. Each stripe column contains 4 samples and JPEG2000 limits the number of stripe columns in any code-block to at most 2^{10}. Thus, sign and magnitude bit-planes require 1K×4-bit memories. We consider an implementation in which the magnitude bit-plane memory is double buffered, as indicated in the figure.

A separate internal memory maintains a set of four binary state variables for each location in the code-block. The significance state variable, $\sigma[\mathbf{j}]$, is central to the bit-plane coding operations described in Section 3.3. The sign state variable, $\hat{\chi}[\mathbf{j}]$, contains the sign bit of any sample which has become significant. This is transferred from the sign bit memory at the appropriate point during the coding process. $\pi[\mathbf{j}]$ is a coding pass membership state variable. It holds 1 if location $\mathbf{j}$ has been included in the significance propagation pass of the current bit-plane. Finally, $\overleftarrow{\sigma}[\mathbf{j}]$ is a delayed version of $\sigma[\mathbf{j}]$; it is initialized to 0 and

Table 3: Useful statistics for estimating the throughput of block encoding and decoding hardware.

Bit-rate	Z_{avg}	R_{sym}	R_{empty}
0.25 bps	2.5	0.50	0.33
0.5 bps	4.5	0.93	0.59
1.0 bps	7.3	1.66	0.94
2.0 bps	11.1	2.89	1.40
lossless	18.4	5.77	2.24

toggled to 1 after the first magnitude refinement operation for sample location $\mathbf{j}$. The value of $\overleftarrow{\sigma}[\mathbf{j}]$ tells us whether or not $v^{(p+1)}[\mathbf{j}] > 1$, when determining the magnitude refinement context in accordance with equation (2).

The contents of the state memory are initialized to 0 and updated as the coding proceeds. In order to determine coding pass membership and form coding contexts for the locations in any given stripe column, we must have access to state variables within a 6×3 window. This window consists of three consecutive columns from the current stripe, from the last row of the preceding stripe and from the first row of the following stripe. The window is managed by a type of shift register, which reduces the number of internal memory accesses. The structure of the shift register should be clear from Figure 9. As the coding proceeds, the elements in this shift register which represent state variables for the current stripe column may be modified.

Although we discuss only the encoder here, a decoder may be implemented in almost identical fashion. In particular, the state variables assume identical values in the encoder and decoder at each coding step. To estimate the throughput of an encoder or decoder implementation, we shall assume that the MQ coder is able to process one symbol per clock cycle. This is not an unreasonable assumption, considering the simplicity of the coding steps described in Section 6.1. This suggests that the average throughput is upper bounded by $1/R_{\text{sym}}$ samples per clock cycle, where R_{sym} is the average number of binary events coded by the MQ coder per image sample. We shall presently consider factors which may prevent us from achieving such a throughput.

Table 3 provides observed values for the symbol rate, R_{sym}, taken over the three large photographic test images, "Bike", "Cafe" and "Woman" (see Section 5). These results are obtained using a reversible 5/3 DWT so that lossless compression is achieved when the code-block bit-streams are not truncated. Strictly speaking, the non-lossless results are directly applicable only for decoding, since the encoder must generally process more coding passes than are usually included in the final bit-stream. These results suggest a maximum average throughput for lossless encoding or decoding of 1/5.8 samples per clock cycle.

To achieve this maximum throughput, the MQ coder must be kept continuously active. In particular, clock cycles cannot be wasted in processing sample locations which do not belong to the current coding pass. For example, if each

sample location of each stripe column required a single clock cycle per coding pass, the average throughput would be limited to $1/Z_{\text{avg}}$ samples per clock cycle, where Z_{avg} is the average number of coding passes per code-block. The observed statistics reported in Table 3 suggest that in such an implementation approximately 75% of the clock cycles would be wasted. The run mode described in Section 3.3.2 also commonly dispatches an entire stripe column with a single symbol.

A more efficient implementation is possible, e.g., by performing multiple tests concurrently to identify the next member location within a single stripe column. In this case, each clock cycle would process the next unprocessed sample location which actually belongs to the current coding pass, so long as the stripe column contains such a location. A clock cycle need only be wasted when a stripe column is "empty," meaning that none of its sample locations belong to the current coding pass. The last column in Table 3 provides observed statistics for the average number of empty stripe columns, R_{empty}, expressed as a fraction of the number of image samples. An enhanced implementation of the form described above should be able to achieve an average throughput of $1/(R_{\text{sym}} + R_{\text{empty}})$ samples per clock cycle. Thus, for truly lossless compression or decompression, an average throughput of 1/8 samples/clock appears to be quite realistic, while much higher throughputs can be achieved at lower bitrates. A reduction of R_{empty} by processing multiple stripe columns together would cause an increase in the peak access bandwidths that internal memories would need to support.

It is instructive to consider the importance of the stripe oriented scanning pattern in Figure 6 to the block coding architecture outlined above. In general, larger stripe heights allow for reduced internal memory access bandwidth. This is because access to the rows above and below each stripe (for context formation) may be amortized over a larger stripe height. On the other hand, larger stripe heights also imply larger registers and more complex coding logic. The JPEG committee adopted stripes of height 4 as a compromise between these extremes. This particular stripe height also has subtle advantages for at least some efficient software implementations, such as that used by the JPEG2000 verification model software. For a detailed discussion of these issues, as well as a number of interesting enhancements to the hardware architecture suggested above, the reader is referred to [8].

6.4 Buffering Resources

The main elements in an implementation of the JPEG2000 standard are the DWT and block coder. Figure 10 illustrates a possible interaction between these elements and the application. The compressor and decompressor have similar properties, with the data flows reversed. We assume that the images involved are large enough to rule out buffering of a single line of image samples in internal memory. Thus, we will consider architectures in which the required internal memory size is independent of the image dimensions. We also assume that the application supplies (or consumes) image samples line by line.

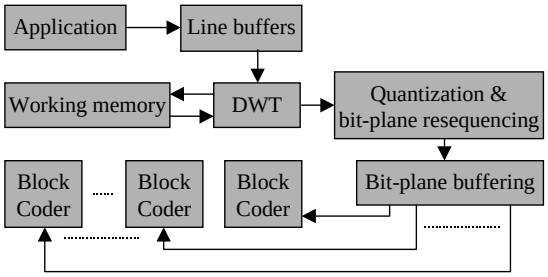

Figure 10: Interaction between DWT and coding sub-systems in a JPEG2000 compressor. Decompressor is similar, with data flow directions reversed.

Each DWT stage is essentially a type of filtering operation and may be implemented incrementally as image lines appear (or are consumed), using well established hardware implementation techniques. Successive DWT stages may be pipelined in various ways so that there is never any need to buffer the entire image or any comparable quantity. For details and analyses of various DWT implementation strategies, the reader is referred to [15, 8]. For our present purposes, it is sufficient to make the following summary observations.

In general, memory size may be traded for memory bandwidth (between on-chip and external memory) by adjusting the number of image sample lines which are processed together. As an example, it is possible to implement the DWT with the CDF 9/7 wavelet kernels using approximately 20 to 80 image lines of working memory. Assuming 8-bit image samples, the upper end of this range allows for external memory bandwidths as low as 2.4 byte transactions per image sample, including the cost of 2 bytes/sample associated with writing each original image sample to memory and subsequently retrieving it[7].

The memory and bandwidth figures quoted above do not include the cost of buffering quantized subband samples in memory prior to coding (or buffering decoded quantization indices in memory prior to the inverse DWT). In some designs it might be possible to avoid such intermediate buffering of quantized subband samples. In the present discussion, however, we shall assume that the system must support the costs associated with such buffering and we endeavour to provide useful estimates for the average values of these costs.

The block coder architecture suggested in Section 6.3 expects to read or write quantized subband samples bit-plane by bit-plane. To make this possible, the subband samples produced by the DWT may be immediately quantized, assembled on-chip into chunks (e.g., 64 samples at a time) and then written out bit-plane by bit-plane to external memory for later coding. This is identified as "bit-plane resequencing" in Figure 10. A dual process may be used for decompression.

[7] In some cases, the application may be prepared to supply or consume image samples in exactly the same order as that required by the DWT implementation, in which case the 2 byte/sample transaction cost might be eliminated.

Table 4: Average buffered bits per subband sample and corresponding buffer memory bandwidth, expressed in byte transactions per sample.

bit-rate:	0.25 bps	0.5 bps	1.0 bps	2.0 bps	lossless
bits/sample:	0.79	1.43	2.46	3.79	5.99
B_{coder}:	0.20	0.36	0.61	0.95	1.50

Bit-plane resequencing has a positive impact on external memory bandwidth. Without such a device, the buffering cost of a similar algorithm is estimated in [22], based on the assumption that each subband sample can be accurately represented with at most twice as many bits as the original image samples. Noting that there are 3 subbands in almost all DWT levels and the width of subbands at level d is 2^{-d} times the width of the original image lines, the above assumption on the precision of the subband samples implies a bound of

$$S_{\text{coder}} \lesssim 2J_1 \sum_{d=1}^{\infty} 3 \cdot 2^{-d} = 6J_1 \text{ image lines}$$

on the subband buffering memory size, where J_1 is the code-block height. Assuming 2 bytes per subband sample, the external memory bandwidth associated with this buffering is 4 byte transactions/sample. In a practical system, the buffer memory size might need to be further increased (e.g., double buffering) to further decouple the DWT and coding sub-systems.

Fortunately, much lower memory bandwidths (and somewhat smaller buffer sizes) may be realized by exploiting the fact that subband samples are buffered bit-plane by bit-plane. The bit-plane resequencer can determine the actual number of bit-planes which must be buffered for each chunk. In most chunks, most of the more significant magnitude bit-planes are entirely zero (insignificant) and need not be explicitly buffered. Further advantages are available during decompression, since many of the least significant bit-planes may not be decoded, due to truncation of the embedded block bit-streams.

Table 4 indicates the average number of bits per subband sample (including significant magnitude bits and the sign bits of non-zero chunks) which must be written to and retrieved from the bit-plane buffer, assuming a chunk size of 64 samples. These results are obtained under the same conditions as those in Tables 1 and 2. The table also reports the corresponding *average* memory bandwidth, B_{coder}, expressed in byte transactions per sample. Evidently, the average memory bandwidth associated with bit-plane buffering can be quite modest.

7 Mode Variations

The JPEG2000 standard allows for a number of variations on the algorithm described thus far. The mode variations are sufficiently minor that the need to

support all modes may not impose a significant burden on decoder implementations.

7.1 Parallel Execution of Coding Passes

The JPEG2000 block coder lends itself to parallel implementation techniques at a "macroscopic" level, since any number of blocks may be encoded or decoded concurrently. The standard also defines modes which enable the parallel processing of individual coding passes within a code-block. Some of the relevant considerations are discussed in [23].

The key steps which must be taken to enable parallel implementation of the coding passes are:

1. Terminate the MQ codeword at the end of each coding pass[8].
2. Initialize the MQ coder and all 18 probability models at the beginning of each coding pass.

Parallel coding pass implementations are greatly facilitated by the addition of a third modification:

3) Eliminate the dependence of coding steps within any given stripe on the significance or sign of samples in the next stripe.

We refer to this third modification as "stripe-causal" coding. Stripe-causal coding affects context formation only for samples in the last row of a stripe column. The JPEG2000 block coder provides mode switches to achieve each of the three modifications described above, either individually or together. The cost of parallelism is surprisingly low, as little as 0.1 dB on the average (with the largest code-blocks).

In order to take advantage of the opportunities for additional parallelism enabled by this mode, an implementation may perform a number of coding passes in parallel, for a number of code-blocks. In the extreme case, an implementation might perform all coding passes in parallel for each and every code-block. It should be noted, however, that based on the statistics shown in Table 3, most of the coding pass processors are likely to be idle most of the time. Alternatively, a smaller number of coding pass processors, say 8, might be implemented. These processors might need to be invoked multiple times in order to process some code-blocks. The need to synchronize parallel coding pass processors may be the cause of additional idle clock cycles for any given processor.

[8] The reader may wonder why this is not required in the sequential mode described hitherto, since the embedded bit-stream may be truncated at the end of any coding pass. The encoder actually records the length of a prefix of the embedded bit-stream, which is sufficient to decode all included coding passes. The decoder is expected to append $\mathtt{FF}_{\mathrm{h}}$'s to the prefix which it actually receives and the encoder typically exploits this fact to determine a suitable (possibly minimal) prefix length.

In sequential mode, lengths are recorded for each quality layer to which the block makes a non-empty contribution. In the parallel mode, length information is recorded for every coding pass. For a thorough discussion of these matters, the reader is referred to [8].

In addition to the opportunities for parallelism, this mode variation enables the reduction of buffering resources in cases where the application supplies or consumes the image in a line-by-line fashion, by decoupling the number of lines jointly processed by the DWT from the block height. Notice that the savings enabled by bit-plane resequencing in Section 6.4 are based on the statistics presented in Table 3. In systems with limited buffering resources, a design may need to assume a worst case scenario. In the parallel mode described in this section, the coder can process and discard a block-stripe, moving to the next code-block and saving the state of the system (context and MQ-coder states, as well as boundary conditions for determining future contexts). The saved state is then retrieved to be used for processing the next stripe in the code-block, as soon as this stripe is supplied by the transform.

In this way, there is no need to buffer entire code-blocks of quantized subband samples in external memory. In fact, by tightly coupling the implementations of the block coder and DWT, intermediate buffering of quantized subband samples may be eliminated altogether [15]. Moreover, this may be achieved without imposing unreasonable constraints on number of lines processed together by the DWT[9]. Of course, saving the coder state information itself imposes a penalty in terms of memory size and memory bandwidth. The possibility of eliminating intermediate buffering of subband samples may be of interest in applications where peak (rather than average) memory bandwidth is an important resource. While the parallel mode is not necessarily advocated as an improvement on memory bandwidth (unless the storage of coder states can be done in internal memory), it clearly provides more flexibility in managing the trade-offs between memory size, memory bandwidth and internal complexity.

7.2 Lazy Coding

As one might expect, symbol probabilities tend to be substantially less skewed in the less significant bit-planes. As a result, little benefit is generally derived from the use of arithmetic coding in the significance propagation, $\mathcal{P}_1^p$, and magnitude refinement, $\mathcal{P}_2^p$, coding passes for $p < K - 4$. The coder provides a mode for bypassing the MQ coder altogether in these coding passes, which can result in significant speedup for software implementations at very bit-rates; it can also reduce the complexity of a parallel coding pass implementation, as discussed above. For most natural images, this mode appears to have negligible effect on compression efficiency. On the other hand, artificial imagery, including text, graphics and compound documents tend to suffer more significantly.

8 Summary

JPEG2000 is an advanced image compression standard which incorporates and emphasizes many features not found in earlier compression standards. Many

[9] It is sufficient for the DWT to produce or consume subband sample lines in multiples of 4, the stripe height.

of these features are imparted by independent, embedded coding of individual blocks of subband samples. In this paper, we have described the embedded block coding algorithm, its various advantages, indicative compression performance and some of its implications for implementation complexity.

There is no doubt that the JPEG2000 standard is substantially more complex than the baseline sequential JPEG algorithm, both from a conceptual and an implementation standpoint. On the other hand, efficient implementations of the algorithm in hardware and software are not beyond reach. JPEG2000 combines state-of-the-art compression performance, with a very rich set of features, which may help to usher in a new generation of imaging applications.

References

[1] ISO/IEC 15444-1: JPEG2000 image coding system (2000).

[2] ISO/IEC 10918-1, ITU Recommendation T.81: Digital compression and coding of continuous tone still images - Requirements and guidelines (September 1993).

[3] J. Shapiro, An embedded hierarchical image coder using zerotrees of wavelet coefficients, Proc. IEEE Data Compression Conf. (Snowbird) (1993) 214–223.

[4] A. Said, W. Pearlman, A new, fast and efficient image codec based on set partitioning in hierarchical trees, IEEE Trans. Circ. Syst. for Video Tech. (1996) 243–250.

[5] D. Taubman, High performance scalable image compression with EBCOT, IEEE Trans. Image Proc. 9 (7) (2000) 1158–1170.

[6] E. Ordentlich, M. Weinberger, G. Seroussi, A low-complexity modeling approach for embedded coding of wavelet coefficients, Proc. IEEE Data Compression Conf. (Snowbird) (1998) 408–417.

[7] J. Li, S. Lei, Rate-distortion optimized embedding, Proc. Picture Coding Symposium, Berlin (1997) 201–206.

[8] D. Taubman, M. Marcellin, JPEG2000: Image Compression Fundamentals, Practice and Standards, Kluwer Academic Publishers, 2001.

[9] D. Taubman, A. Zakhor, A common framework for rate and distortion based scaling of highly scalable compressed video, IEEE Trans. Circ. Syst. for Video Tech. 6 (4) (1996) 329–354.

[10] D. Taubman, A. Zakhor, Multi-rate 3-d subband coding of video, IEEE Trans. Image Proc. 3 (5) (1994) 572–588.

[11] A. Zandi, J. Allen, E. Schwartz, M. Boliek, CREW: Compression with reversible embedded wavelets, Proc. IEEE Data Compression Conf. (Snowbird) (1995) 212–221.

[12] E. Ordentlich, M. Weinberger, G. Seroussi, On modeling and ordering for embedded image coding, in: Proc. 2000 IEEE Int. Symp. Inf. Theory (ISIT'00), Sorrento, Italy, 2000, p. 297.

[13] F. Sheng, A. Bilgin, P. Sementilli, M. Marcellin, Lossy and lossless image compression using reversible integer wavelet transforms, Proc. IEEE Int. Conf. Image Proc. (1998) 876–880.

[14] X. Wu, High order context modeling and embedded conditional entropy coding of wavelet coefficients for image compression, in: Proc. of the 31st Asilomar Conf. on Signals, Systems, and Computers, Asilomar, USA, 1997, pp. 1378–1382.

[15] M. Marcellin, T. Flohr, A. Bilgin, D. Taubman, E. Ordentlich, M. Weinberger, G. Seroussi, C. Chrysafis, T. Fisher, B. Banister, M. Rabbani, R. Joshi, Reduced complexity entropy coding, Tech. Rep. N1312, ISO/IEC JTC1/SC29/WG1 (June 1999).

[16] J. Li, P. Cheng, C.-C. J. Kuo, On the improvements of embedded zerotree wavelet (EZW) coding, Proc. SPIE: Visual Comm. and Image Proc. (Taipei) 2601 (1995) 1490–1501.

[17] A. Cohen, I. Daubechies, J.-C. Feauveau, Biorthogonal bases of compactly supported wavelets, Communications on Pure and Appl. Math. 45 (5) (1992) 485–560.

[18] D. Santa-Cruz, T. Ebrahimi, An analytical study of JPEG2000 functionalities, Proc. IEEE Int. Conf. Image Proc. .

[19] W. Pennebaker, J. Mitchell, G. Langdon, R. Arps, An overview of the basic principles of the q-coder adaptive binary arithmetic coder, IBM J. Res. Develop. 32 (6) (1988) 717–726.

[20] W. Pennebaker, J. Mitchell, JPEG: Still Image Data Compression Standard, Van Nostrand Reinhold, New York, 1992.

[21] D. Taubman, E. Ordentlich, M. Weinberger, G. Seroussi, I. Ueno, F. Ono, Embedded block coding in JPEG2000, Proc. IEEE Int. Conf. Image Proc. .

[22] D. Taubman, EBCOT: Embedded block coding with optimized truncation, Tech. Rep. N1020R, ISO/IEC JTC1/SC29/WG1 (October 1998).

[23] E. Ordentlich, D. Taubman, M. Weinberger, G. Seroussi, M. Marcellin, Memory efficient scalable line-based image coding, Proc. IEEE Data Compression Conf. (Snowbird) (1999) 218–227.