AN EFFICIENT PACKETIZATION ALGORITHM FOR JPEG2000

Wei Yu, Jason Fritts

Washington University
Computer Science Department
St. Louis, MO 63130

Fangting Sun

Iowa State University
Computer Science Department
Ames, IA 50011

ABSTRACT

This paper presents an efficient algorithm for post compression optimal rate allocation and packetization within JPEG2000 encoding. JPEG2000, the new ISO/ITU-T standard for still image coding, has been shown to provide superior coding efficiency to the previous standard, JPEG. However, the added efficiency of JPEG2000 comes at the cost of increased computational requirements. To improve the computational efficiency of JPEG2000, we propose a new algorithm for JPEG2000 rate allocation and packetization utilizing the *D-Heap* data structure. Implemented in *Jasper* and tested on five reference images, this algorithm provides a speedup for JPEG2000's rate allocation and packetization of 15.9 times on average, and enables an average overall speedup of 33% for JPEG2000 encoding.

1. INTRODUCTION

JPEG2000 [1], the new ISO/ITU-T still-image standard, has been shown to provide superior coding efficiency to the previous standard, JPEG. However, the added efficiency of JPEG2000 comes at the cost of increased execution time. To improve the computational efficiency of JPEG2000, we propose a new algorithm for rate allocation and packetization in JPEG2000 using the *D-Heap* data structure. Using this algorithm, we can substantially decrease execution time for rate allocation and packetization.

JPEG2000 is a wavelet-based image compression standard using the EBCOT (Embedded Block Coding with Optimized Truncation) image compression algorithm [2, 3]. The JPEG2000 encoding process decomposes a still image into a hierarchical organization involving multiple resolution levels, subbands, precincts, and code blocks, as illustrated in Fig. 1. The general structure of the JPEG2000 encoder is shown in Fig. 2. The encoder first translates the separate RGB color components of the image into the corresponding YCbCr color space. Subsequently, the wavelet transform decomposes each component into several *resolution levels*, each containing a series of *subbands* (3 subbands at each level, except for the lowest resolution level, which has 4). The coefficients in each wavelet subband are then quantized and divided into regular arrays of *precincts* and *code blocks* for entropy coding.

Each code block is entropy-coded independently using the recursive probability interval subdivision of Elias coding [4]. Each entropy-coded code block is composed of several *coding passes* (usually called embedded bit-streams) and each coding pass provides a variable quality contribution to the reconstructed image. Following entropy-coding, a post compression rate allocation algorithm selects coding passes from each entropy-coded code block

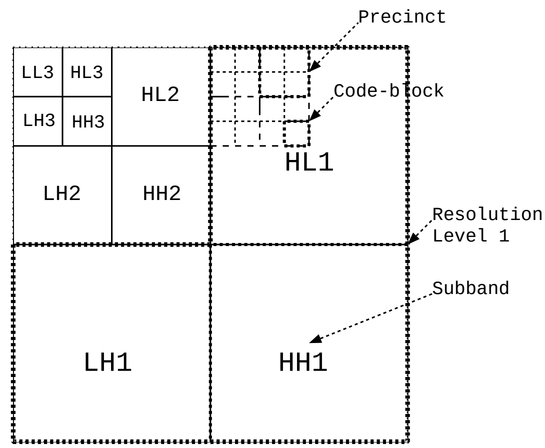

Fig. 1. Partitioning a wavelet-transformed component into resolution levels, subbands, precincts and code blocks

so that an optimal allocation, minimizing image distortion according to the desired bitrate, is achieved. The selected coding passes from each code block are packetized, and all the data packets are assembled into the final coding stream. Each data packet is just a collection of coding passes in the final coding stream, and is composed of two parts: header and body. The header indicates which coding passes are included in this packet, and the body contains the actual data from the included coding passes. The packetization procedure imposes a particular organization on coding pass data, commonly referred to as the progression mode, which facilitates many of the desired codec features, including rate scalability and progressive recovery by fidelity or by resolution [4].

The remainder of this paper is organized as follows. In section 2, we introduce the optimal rate allocation scheme used in JPEG2000. In section 3, we propose an efficient algorithm and implementation for this scheme. Section 4 evaluates the performance of our algorithm. Finally, section 5 concludes the paper and discusses future work.

2. RATE DISTORTION OPTIMIZATION

This section gives a concise overview of the optimal rate allocation scheme used by JPEG2000. A much more thorough discussion of the rate distortion optimization component of the EBCOT algorithm can be found in [3].

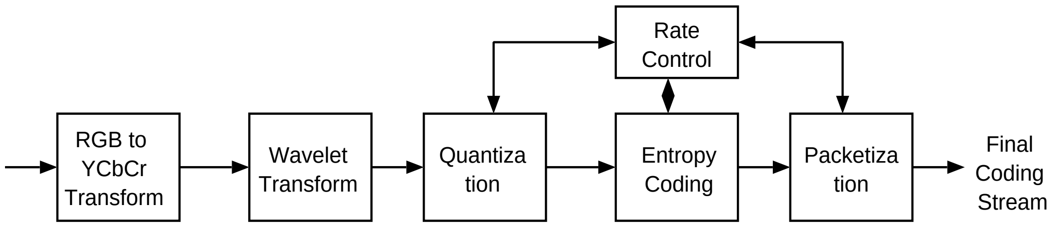

Fig. 2. Structure of JPEG2000 Encoder

As illustrated in Figure 1, each component of the wavelet transformed image is partitioned into a collection of relatively small code blocks, B_i, whose entropy-coded embedded bit-streams may be truncated to rates, R_i^n, according to the specified maximum bitrate for the final coding stream. The contribution from B_i to distortion in the reconstructed image is denoted D_i^n, for each truncation point, n. Here the relevant distortion metric can be assumed to be additive, i.e.,

$$D = \sum_i D_i^{n_i} \tag{1}$$

where D represents the overall image distortion and n_i denotes the truncation point selected for code block B_i. Usually, an additive distortion metric which approximates Mean Squared Error (MSE) is used by setting

$$\hat{D}_i^n = w_{b_i}^2 \sum_{k \in B_i} (\hat{s}_i^n[\vec{k}] - \hat{s}_i[\vec{k}])^2 \tag{2}$$

Here, $\hat{s}_i[\vec{k}]$ denotes the two-dimensional (2-D) sequences of subband samples in code block B_i, $\hat{s}_i^n[\vec{k}]$ denotes the quantized representation of these samples associated with truncation point n, and w_{b_i} denotes the coefficient weighting for the subband, b_i, to which code block B_i belongs.

To minimize distortion subject to a specified maximum bitrate, R^{max}, the rate allocation algorithm must find the optimal set of truncation points that satisfies the following equation:

$$R^{max} \geq R = \sum_i R_i^{n_i} \tag{3}$$

The problem of selecting the optimal truncation points to satisfy Equation 3, can be solved using a generalized Lagrange multiplier method [5]. Any set of truncation points, $\{n_i^\lambda\}$, which minimizes Equation 4 for some λ

$$(D(\lambda) + \lambda R(\lambda)) = \sum_i (D_i^{n_i^\lambda} + \lambda R_i^{n_i^\lambda}) \tag{4}$$

is optimal in the sense that the distortion cannot be reduced without also increasing the overall rate and vice-versa.

The determination of the optimal truncation points, n_i^λ, for any given λ, may be performed very efficiently based on a small amount of summary information collected during the generation of each code block's embedded bit-stream. It is clear that, for each code block B_i, it is a separate minimization problem. A simple algorithm to find the truncation point, n_i^λ, which minimizes $\Sigma_i(D_i^{n_i^\lambda} + \lambda R_i^{n_i^\lambda})$, is as follows:

- Initialize n_i^λ = 0;
- For j = 1, 2, 3, . . .
  - Set $\Delta R_i^j = R_i^j - R_i^{n_i^\lambda}$ and $\Delta D_i^j = D_i^{n_i^\lambda} - D_i^j$;
  - If $\Delta D_i^j / \Delta R_i^j > \lambda$ then update $n_i^\lambda = j$;

We first find the subset of feasible truncation points, N_i. Let $j_1 < j_2 <$ ... be an enumeration of these feasible truncation points and let the corresponding distortion-rate $slopes$ be given by $S_i^{j_k} = (\Delta D_i^j / \Delta R_i^j)$ where $\Delta R_i^{j_k} = R_i^{j_k} - R_i^{j_{k-1}}$ and $\Delta D_i^{j_k} = D_i^{j_k} - D_i^{j_{k-1}}$. These slopes must be strictly decreasing, for if $S_i^{j_{k+1}} \geq S_i^{j_k}$, then the truncation point, j_k, could never be selected by the above algorithm, regardless of the value of λ, contradicting the fact that N_i is the feasible set of truncation points.

When restricted to a set of truncation points whose slopes are strictly decreasing, the above algorithm reduces to the trivial selection $n_i^\lambda = max\{j_k \in N_i | S_i^{j_k} > \lambda\}$ so that each such point must be a valid candidate for some value of λ. It follows that N_i is the largest set of truncation points for which the corresponding distortion-rate slopes are strictly decreasing. This unique set may be determined using a conventional convex hull analysis.

3. HEAP-BASED PACKETIZATION ALGORITHM

We propose a new algorithm for more efficiently implementing post compression rate allocation and packetization in JPEG2000. This algorithm implements the optimal rate allocation scheme in JPEG2000 [3], as discussed above in section 2, using the Minimum Spanning Tree Algorithm [6] and the *D-Heap* data structure.

In JPEG2000 encoding, the subset of feasible truncation points, N_i, can be determined immediately after code block B_i has been entropy-coded. Once N_i has been determined, the next process is post compression rate allocation and packetization. During rate allocation and packetization, code streams may be generated as either resolution-scalable or quality-scalable. The post compression rate allocation and packetization schemes for generating these two progressive scalability modes varies slightly. This paper mainly focuses on generating quality-scalable code stream, although the principle for generation resolution-scalable code streams is the same and the implementation is similar.

For rate allocation and packetization of quality-scalable code streams, the final coding stream is composed of one or more quality layers, numbered from 0 to $L-1$, where $L > 0$. Each entropy-coded coding pass is either assigned to one of the L layers or discarded. The coding passes containing the most important data are included in the lower layers, while the coding passes associated with finer details are included in the higher layers.

The post compression rate allocation and packetization algorithm we implemented assigns segments from each code block to quality layers, where we define a *segment* as the set of coding passes between two consecutive feasible truncation points. The algorithm is based on the *D-Heap* data structure. Each node in the heap represents a separate entropy-coded code block. The key of a node is the distortion-rate slope of the first unmarked coding segment of this code block, where a segment is marked if and only if it has been selected and assigned to a quality layer. The nodes are sorted according to their key values and the root node has the maximum key value, which means that it has the largest contribution to the distortion among all the unmarked segments in this code block. In each step, only the root node is processed, at which time its first unmarked coding segment is marked and the corresponding segment is assigned a quality layer. Then this node is updated and the new key is generated according to the next unmarked segment in this code block which has the largest key value. The details of the algorithm are given below.

procedure packetization (**set** of cblk s, **int** layernum);
int cumrate, curlayer; **heap** h;
mapping rate : layernumber $\rightarrow$ layerrate;
$h := makeheap(s)$;
$curlayer := 0$;
while *($curlayer < layernum$)* **do**
  $cumrate := 0$;
  while *($cumrate < rate(curlayer)$)* **do**
    $cumrate := cumrate + length(root(h))$;
    mark the first unmarked segment of root(h);
    update the key value of root(h);
    $siftdown(root(h), key(root(h)), h)$;
  endwhile
  encodepackets(the segments marked in this loop);
  $curlayer := curlayer + 1$;
endwhile

heap function makeheap (**set** of cblk s);
int j; **heap** h;
$h := \emptyset$;
for $(i \in s) j := |h| + 1; h(j) := i$; **rof**;
$j = ((|h| - 1)/d)$;
while *($j > 0$)* **do**
  $siftdown(h(j), key(h(j)), h)$;
  $j := j - 1$;
endwhile
return h;

procedure siftdown (cblk i, **int** x, modifies **heap** h);
int c;
$c := maxchild(x, h)$;
while ($c \neq Null$) *and* ($key(h(c)) > key(i)$) **do**
  $h(x) := h(c); x := c; c := maxchild(x, h)$;
endwhile
$h(x) := i$;

int function maxchild (int x, heap h);
int i, maxc;
$maxc := d(x - 1) + 2$;
if ($maxc > |h|$) **return** 0; **fi**;
$i := maxc + 1$;
while $i \in min\{|h|, dx + 1\}$ **do**
  if ($key(h(i)) < key(h(maxc))$) maxc := i; **fi**;
  $i := i + 1$;
endwhile
return maxc;

Regarding this algorithm, $length(root(h)$ denotes the length of the first unmarked segment that will be marked in the inner *while* loop; $layernum$ indicates the number of quality layers, and the number of bits per quality layer, in the final code stream. Once the number of bits for a quality layer has been satisfied, the $encodepackets$ procedure is called to packetized this quality layer. Also, we use $d = 2$.

The computational complexity of the proposed algorithm is $O(nd\log_d n)$. Each execution of *siftdown* takes time $O(d\log_d n)$, where n denotes the number of code blocks in this image. The running time for *makeheap* is $O(f)$ where

$$f(n) = \frac{n}{d}d + \frac{n}{d^2}2d + \frac{n}{d^3}3d + ...$$

which is $O(n)$. Assuming the bit-depth per component of the original image is 8 bits, each code block has no more 22 coding passes (i.e. 3*(*b*-1)+1, where *b*=8), so there is a maximum of 22 segments per code block. Since each segment is processed once at most, the total running time is $O(22 * nd\log_d n)$, which is $O(nd\log_d n)$.

To further improve the performance of the post compression rate allocation packetization procedure, several modifications are made. First, only pointers to these entropy-coded code blocks are stored in the heap, which reduces heap traversal time. Second, the heap is implemented using a vector, which helps reduce cache miss rates. Third, we consider the amount of header overhead when packaging the selected segments, using $rate(curlayer) - \Delta$ instead of simply $rate(curlayer)$ as the bound on packet size, where Δ is estimated using statistical analysis. For our implementation, we let $\Delta = 5\% * rate(curlayer)$ for all packets except the final packet in the last layer, in which we allow an additional 200 bytes for the final packetization overhead. Finally, we delete a code block from the heap once all its segments have been marked (i.e. allocated), which helps further reduce execution time.

4. PERFORMANCE EVALUATIONS AND COMPARISON

We evaluated the performance of our proposed algorithm in two areas: execution speed and allocation precision. We used *Jasper* [7], one implementation of the JPEG2000 standard written in C, as our baseline JPEG2000 codec, and modified the rate allocation and packetization method to use the proposed algorithm, as discussed above in section 3.

Evaluation experiments of both the original Jasper codec and the Jasper codec with the *D-Heap*-based rate allocation and packetization algorithm were performed using five images, the characteristics of which are given in Table 1 (*bpp* indicates "bits per pixel" per component, and *comp.* indicates the number of components). The images were all encoded using a 10:1 compression ratio, and all quality layers had the same length. Finally, execution times were measured on a 733 MHz Pentium III running Linux, and all experiments used the Daubechies integer 5/3 reversible wavelet filter [8].

Table 1. Image Description.

Name	Resolution	bpp	Comp.
Monarch	768x512	8	3
Fruit	512x512	8	3
Baboon	512x512	8	3
Lena	512x512	8	3
Mona-Lisa	605x790	8	3

Table 2. Execution Speed Comparison.

Image	$L = 1$	$L = 5$	$L = 10$
Monarch	12.9, 1.1	14.5, 1.23	14.7, 1.31
Fruit	12.1, 1.13	12.6, 1.23	13.2, 1.27
Baboon	14.6, 1.19	15.1, 1.25	19.4, 1.33
Lena	13.8, 1.16	13.9, 1.23	13.6, 1.28
Mona-Lisa	12.8, 1.18	16.2, 1.32	18.4, 1.46
Average	13.24, 1.15	14.46, 1.25	15.86, 1.33

Table 2 indicates the execution time results. In this table, L indicates the number of quality layers generated when encoding the images. In each cell, the first value is the execution speedup of the *D-Heap*-based rate allocation and packetization procedure over the original *Jasper* implementation, and the second value gives the overall execution speedup of the JPEG2000 encoding using the *D-Heap*-based rate allocation and packetization algorithm over the original *Jasper* implementation. Each speedup value represents an average of 10 runs of the program for that image and number of quality layers. The results show that the speedup of the rate allocation and packetization procedure varies across different numbers of quality layers from 13.2 to 15.9 on average, and the overall speedup varies similarly from 1.15 to 1.33 in average, with the speedup increasing as the number of quality layers increases.

Table 3 presents the average allocation precision measurements of our algorithm. In this experiment, there were ten quality layers in each of the final coding streams. Once again, the compression ratio was 10:1 and all quality layers were of equal size. Each entry in Table 3 indicates the deviation in length of each quality layer compared with its specified length. Overall, the deviation is very minor (less than 1% for most quality layers). The results show that the rate allocation precision of our algorithm is excellent.

5. CONCLUSION AND FUTURE WORK

In this paper we designed and implemented an efficient rate allocation and packetization algorithm for JPEG2000 based on the *D-Heap* data structure. The results show that our implementation is much faster, with speedups up to 15.9 on average for the rate allocation and packetization procedure, giving overall encoding execution times of 33rate allocation precision is excellent, with the allocation precision typically deviating by less than 1%. Finally, while this implementation focused on generating quality-scalable coded bitstreams, the same principle is effectively the same for coding resolution-scalable bitstreams.

Table 3. Precision Evaluation.

layer	monarch	fruit	baboon	lena	mona-lisa
0	-3.2%	-2.2%	-4.1%	-4.2%	-3.2%
1	-0.6%	+1.9%	+1.5%	-0.36%	-1.0%
2	+0.3%	-0.05%	-1.5%	+1.3%	+1.3%
3	+0.35%	+0.06%	+3.5%	-0.53%	-0.83%
4	+0.10%	+0.28%	-3.5%	-0.59%	+1.2%
5	-0.76%	+0.79%	+2.3%	+2.4%	-1.4%
6	+0.50%	+0.39%	-1.7%	-2.1%	+0.87%
7	+0.44%	-0.83%	-0.3%	+0.63%	-0.32%
8	-0.69%	-0.49%	+1.9%	-0.53%	+1.0%
9	+0.88%	-0.52%	-0.9%	-0.91%	-1.0%
Total	-0.28%	-0.44%	-0.3%	-0.49%	-0.34%

When designing the *D-Heap*-based rate allocation and packetization algorithm, we found that when the final compression ratio is high, many of the coding passes in each code block are never used. Since the entropy coding is one of the most time consuming parts of JPEG2000 encoding, we plan to investigate new rate allocation and packetization algorithms, which can direct the encoding of individual coding passes for code blocks such that another pass is generated if and only if it is necessary. Using this algorithm, we anticipate significantly higher speedups will be obtained.

6. REFERENCES

[1] ISO/IEC, "ISO/IEC 14495-1:1999: Information technology — Lossless and near-lossless compression of continuous-tone still images: Baseline," 1999.

[2] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image Coding Using Wavelet Transform," *IEEE Transactions on Image Processing*, vol. 1, no. 2, pp. 205–220, 1992.

[3] D. Taubman, "High Performance Scalable Image Compression with EBCOT," *IEEE Transactions on Image Processing*, vol. 9, no. 7, pp. 1158–1170, 2000.

[4] C. Christopoulos, A. Skodras, and T. Ebrahimi, "The JPEG2000 Still Image Coding System: An Overview," *IEEE Transactions on Consumer Electronics*, vol. 46, no. 4, pp. 1103–1127, November 2000.

[5] H. Everett, "Generalized Lagrange Multiplier Method for Solving Problems of Optimum Allocation of Resources," *Operations Research*, vol. 11, pp. 399–417, 1963.

[6] R. E. Tarjan, *Data Structures and Network Algorithms*, CBMS 44. SIAM, Philadelphia, Pennsylvania, 1983.

[7] M. D. Adams, "The JPEG-2000 Still Image Compression Standard," http://.ece.ubc.ca/ mdadams.

[8] A. R. Calderbank, I. Daubechies, W. Sweldens, and B.-L. Yeo, "Wavelet Transforms that Map Integers to Integers," *Applied and Computational Harmonic Analysis*, vol. 5, no. 3, pp. 332–369, July 1998.