HIGH THROUGHPUT JPEG 2000: NEW ALGORITHMS AND OPPORTUNITIES

D. Taubman, A. Naman and R. Mathew

UNSW Sydney, Australia

ABSTRACT

This paper describes a drop-in replacement for the JPEG 2000 (J2K) block coder that offers exceptionally high throughput, with a small reduction in coding efficiency, while retaining all features of J2K except for quality scalability. Throughputs on the order of 10x or more are achievable relative to J2K. We coin the term FBCOT (Fast Block Coding with Optimized Truncation) for the overall proposed algorithm. Truly reversible transcoding between J2K and FBCOT bit-streams is supported on a block-by-block basis, enabling systems in which the efficiency and scalability of J2K can be combined with the high throughput benefits of FBCOT. Based on the outstanding performance of FBCOT, a new standardization activity has been launched within the ISO/IEC working group known as JPEG.

INTRODUCTION

The JPEG 2000 (J2K) image compression standard provides high compression efficiency along with a unique set of features, enabling extraction of resolutions and spatial regions of interest directly from the coded representation. These features render it particularly valuable for interactive and cloud-based content distribution applications. J2K is the basis of Digital Cinema and is used more widely within the video entertainment, broadcast and archival industries. We can expect to see J2K increasingly used in the future, since it offers highly efficient interactive access to content that is too large to consume or view all at once, including VR content. This is enabled by the JPIP standard (J2K Part 9) for interactive transport of image/video content based on user defined windows of interest. For a comprehensive treatment of the JPEG 2000 standard and an introduction to JPIP, the reader is referred to Taubman and Marcellin (1) and Taubman and Prandolini (2).

The main obstacle to more widespread adoption of J2K in entertainment and broadcast sectors has been the relatively high computational complexity of its underlying block coding algorithm. This paper presents a drop-in replacement for the block coding algorithm that is capable of very high throughputs and energy efficient deployment in software and hardware. Moreover, the new coding algorithm supports truly reversible transcoding to or from any J2K code-stream, on a block-by-block basis. This opens up a range of opportunities for new media compression solutions and systems.

Early evidence based on the algorithm described in this paper has inspired the creation of a new activity within the ISO/IEC JTC1/SC29 technical working group WG1 (a.k.a. JPEG). This activity, known as High Throughput JPEG 2000 (HT), is expected to culminate in 2018 with Part 15 of the JPEG 2000 family of standards after a competitive process.

KEY CONCEPTS FROM THE JPEG 2000 BLOCK CODING ALGORITHM

J2K is based on the algorithm known as EBCOT (Embedded Block Coding with Optimized Truncation), Taubman (3). After multi-component decorrelation transformation, a Discrete Wavelet Transform (DWT) decomposes source image data into subband images, which are separately encoded; this provides the energy compaction required for efficient compression, along with resolution scalable access to the coded content. Each subband image is further partitioned into code-blocks, typically of size 64x64 or 32x32, and each code-block is subjected to a fractional bit-plane coding process. This stage endows the compressed representation with quality scalable and region-of-interest based accessibility attributes that are especially important for efficient and responsive interactive access to large media sources; it also provides a high level of macroscopic parallelism to the algorithm, since blocks can be encoded/decoded in parallel. Each block bit-stream can independently be truncated at a rich set of truncation points (3 per bit-plane), allowing rate-distortion optimization decisions to be made after the encoding, in what is known as the PCRD-opt (Post-Compression Rate-Distortion optimization) phase. An outer set of (Tier-2) encoding and packaging tools encapsulates block bit-stream fragments and relevant side information within a stream of so-called J2K packets that form the final code-stream.

To appreciate the innovations described in this paper, it is helpful to briefly review the J2K block coding algorithm. Let $X[\boldsymbol{n}]$ denote the samples within a code-block, indexed by location $\boldsymbol{n} = (n_1, n_2)$, where $0 \le n_1 < W$ represents horizontal position, $0 \le n_2 < H$ denotes vertical position, and *W* and *H* are the code-block's width and height. Let $M_p[\boldsymbol{n}] = \left\lfloor \left| X_p[\boldsymbol{n}] \right| / 2^p \Delta \right\rfloor$ be the sample magnitude, assessed with respect to magnitude bit-plane p, with quantization step size Δ; that is, with *p* LSB's discarded after deadzone quantization. We say that $X[\boldsymbol{n}]$ is "significant" with respect to bit-plane *p* if $M_p[\boldsymbol{n}] \neq 0$. The J2K block coder progressively codes the values $M_p[\boldsymbol{n}]$, starting with $p = P_{max}$ and working down towards the finest bit-plane $p = 0$, where the coarsest bit-plane P_{max} is explicitly communicated as part of the (Tier-2) encoding of side-information during J2K packet construction. Usually, P_{max} is the coarsest bit-plane for which any sample in the block is significant. The block coder makes 3 coding passes per bit-plane, known as "Cleanup," "SigProp" and "MagRef," each of which communicates at most one new magnitude bit and, for newly significant samples, a sign bit.

This multi-pass coding strategy provides a rich set of truncation points, imparting exceptional quality scalability attributes to the representation. While optimization tricks can be used to avoid touching all samples in every coding pass, the throughput of a J2K block coder inevitably declines with increasing quality (number of bit-planes).

FAST BLOCK CODING WITH OPTIMIZED TRUNCATION (FBCOT)

The Fast block coding algorithm described here sacrifices quality scalability for throughput, by encoding multiple bit-planes within a single pass. To enable truly reversible transcoding between J2K and Fast block bit-streams, the Fast block coder also adopts a coding pass structure with Cleanup, SigProp and MagRef coding passes, defined with respect to bit-planes p. Significantly, however, the Cleanup pass associated with bit-plane p fully encodes the magnitudes $M_p[\boldsymbol{n}]$; by contrast the J2K Cleanup pass codes only the significance of samples that are significant in bit-plane p, but not in bit-plane $p + 1$.

Meanwhile, the Fast SigProp and MagRef coding passes code the same incremental refinement information as their J2K counterparts, but using methods that are better suited to high throughput implementation in software. Figure 1 compares the Fast and J2K coding pass structures. In transcoding applications, the refinement passes are required to exactly replicate the adaptive quantization properties of J2K. For content that is directly encoded using the Fast block coder, the refinement passes provide additional truncation points that improve rate control performance, but they are not strictly required.

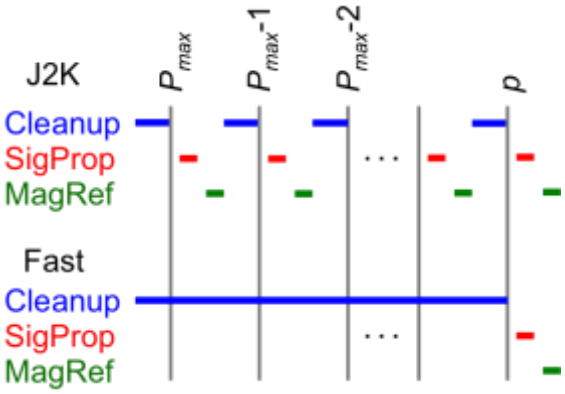

Figure 1 -- coding pass structure

The Fast block coder has no implications for the syntax or Tier-2 coding tools associated with J2K packet construction in JPEG 2000. From a decoder's perspective, at most 3 coding passes are recovered for each code-block. The first is always a Cleanup pass that is characterized by the P_{max} value explained above, but in the Fast variant P_{max} is typically smaller, since the single Cleanup pass includes information from coarser bit-planes.

Within an encoder, multiple bit-planes may be processed for each code-block, yielding a family of coding passes which may subsequently be truncated in a rate-distortion optimal manner, as in the original EBCOT algorithm. However, once the optimal truncation point is found, only the finest Cleanup pass is actually included in the code-stream, along with any subsequent refinement passes, since the finest Fast Cleanup pass subsumes all earlier coding passes. While a J2K block coder must generate all coding passes in sequence, up to some point at which the passes are likely to be discarded by the PCRD-opt stage, an intelligent Fast block coder can start encoding from a later bit-plane, generating a smaller set of (useful) passes as fodder for the PCRD-opt stage. We have coined the term FBCOT for the overall system, involving the Fast block coder and optimized coding pass selection.

Elements of the Fast Cleanup Pass

The Fast Cleanup pass produces a stream of bytes whose length is explicitly signalled to the decoder, known in JPEG 2000 as a "codeword segment." The J2K "RESTART" block coding mode already supports the communication of separate codeword segments for each coding pass via the Tier-2 coding and packet assembly machinery. However, the Fast Cleanup pass actually packs three byte streams into this one codeword segment, as shown in Figure 2. The three byte streams correspond to separate coding processes, which can run in parallel. This is beneficial for software and hardware implementations, increasing concurrency and reducing the number of temporary state variables.

The last two byte streams grow in opposite directions, so that there is no need to separately communicate their lengths. The MagSgn byte stream is harder to bound in length, a priori, so we place this one first, communicating the interface between the two forward growing streams via an 11 bit interface locator word. Each byte stream is separately subjected to a bit stuffing procedure that guarantees no FFh byte will be followed by a byte in the range 90h to FFh – a J2K requirement. Moreover, this condition is also enforced by the termination procedures applied to each byte stream.

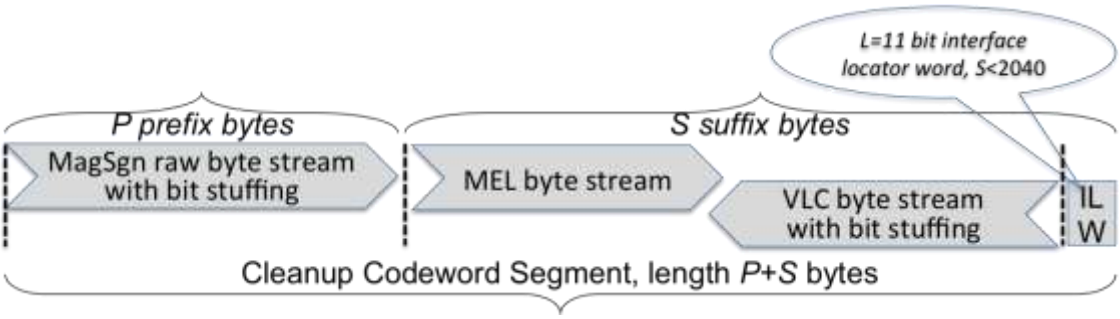

Figure 2 -- triple byte stream codeword segment for the Fast Cleanup pass

The information carried by the three byte streams is as follows:

1. Significant samples are identified using the combination of an adaptive "MELCODE" and a context-adaptive variable length code (VLC). Originally developed by Mitsubishi Electric Labs (MEL), the MELCODE converts symbols to binary digits using a small state machine, with similar properties to arithmetic coding. These digits form the forward growing "MEL byte stream." Meanwhile, the VLC code bits contribute to the backward growing VLC byte stream.
2. For each significant sample, the sign bit, together with 0 or more LSBs of $M_p - 1$, are packed in the MagSgn byte stream as raw bits, subject only to bit-stuffing.
3. The number of magnitude bits that appear for each significant sample is communicated by a separate variable length code, known as the "u-code." The u-code bits are interleaved with significance VLC bits within the VLC byte stream.

Magnitude Exponents

In bit-plane *p*, each sample is assigned a "magnitude exponent" $E_p[\boldsymbol{n}]$ that is defined by

$$E_p[\boldsymbol{n}] = min\left\{E \in \mathbb{N} \middle| M_p[\boldsymbol{n}] - \frac{1}{2} < 2^{E-1}\right\}$$

where $\mathbb{N}$ is the set of natural numbers. For insignificant samples, $E_p[\boldsymbol{n}] = 0$; otherwise, $M_p[\boldsymbol{n}]$>0 and $E_p[\boldsymbol{n}] - 1$ is the number of LSB's from $M_p[\boldsymbol{n}] - 1$ that can be non-zero.

Group Structure and Coding Tools

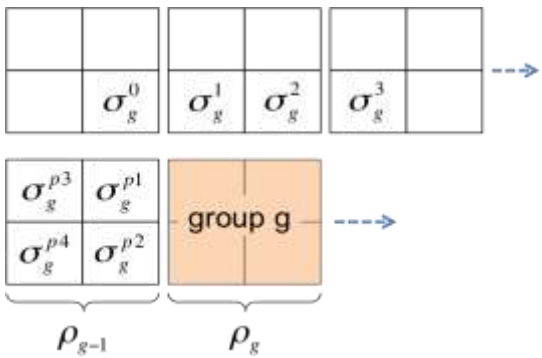

Figure 3 – significance context

Significance is coded in 2x2 groups. Each group *g* has a 4-bit "significance pattern" ρ_g, which is coded with respect to a context that is derived from the significance of neighbouring samples, as illustrated in Figure 3. Eight distinct coding contexts c_g are formed from the significance of these neighbouring samples; moreover, $c_g = 0$ corresponds to the special all-zero context (AZC), in which all samples neighbouring the group are insignificant. For AZC groups (those with $c_g = 0$), a binary group significance symbol σ_g is first coded, indicating whether the entire group is insignificant ($\sigma_g = 0$) or the group has at least one significant sample ($\sigma_g = 1$). This group significance symbol is coded using an adaptive MELCODE that is similar to that employed by the

JPEG-LS standard, but uses a custom state machine with only 13 states. Experiments reveal that this adaptive coding engine has essentially the same efficiency as the MQ arithmetic coder in JPEG 2000, yet encoding/decoding throughput can be much higher due to the small state machine and simple state transition map.

For non-AZC groups, and significant AZC groups, the significance pattern is encoded using 8 VLC tables, one for each context c_g, where each table has 16 codewords, except for the first one ($c_g = 0$), where $\rho_g \neq 0$ is known so that only 15 significance patterns are possible. Codeword lengths are constrained to at most 6 bits to minimize complexity.

For 2x2 groups that are significant, an additional variable length code is used to communicate an upper bound U_g on the value of $E_p[\boldsymbol{n}] - 1$ for each location $\boldsymbol{n}$ in the group. Recall that $E_p[\boldsymbol{n}] - 1$ is the number of LSB's from $M_p[\boldsymbol{n}] - 1$ that can be non-zero, so U_g is a sufficient number of magnitude bits to include in the MagSgn byte stream for each significant sample in group g. If the group has only one significant sample the bound is required to be tight, so the most significant of these $U_g = E_p[\boldsymbol{n}] - 1$ bits is certain to be 1 and not included in the MagSgn byte stream. The value of U_g is represented as $U_g = u_g + \kappa_g$, where κ_g is a predictor that is derived from individual magnitude exponents within the preceding line-pair, while u_g is a non-negative offset that is the subject of variable length coding. Significance VLC bits and u-code bits are interleaved within the VLC byte stream.

What makes it fast?

All elements in the Fast block coder have been carefully selected and parameterized to optimize throughput-performance trade-offs, both in software and hardware deployments. The algorithm is extensively vectorizable in both the encoder and decoder, capable of fully exploiting the benefits of modern instruction sets such as AVX2, NEON, BMI2 and so forth, to the extent that they are available. The triple byte stream structure of the Fast Cleanup pass decouples different stages of the encoding and decoding processes, yielding substantial benefits even for software deployments. Variable length coding technologies have also been carefully selected, so that lookup-based implementations become favourable in software. As with the J2K block coder, each block can be encoded or decoded in parallel, but the Fast block coder processes multiple bit-planes at once, avoiding the need for multiple sequential passes through a code-block's samples.

INTERACTION WITH JPIP

Part-9 of the JPEG 2000 family of standards, known as JPIP, provides a powerful and highly responsive framework for interactive communication of imagery over networks – cf. (2). JPIP provides a rich query language for imagery and metadata, together with streaming media types that can transport just the relevant information to support a dynamic Window of Interest (WOI). The WOI includes region, resolution and image components of interest to the JPIP client. The standard provides effective tools to support lossy transports, pre-emptive requests, stateless and stateful service models, cache synchronization and multiple real or virtual channels, allowing multiple dynamic windows of interest to be served without redundant communication of content. JPIP also supports interactive navigation through video, arbitrary animations and volumetric imagery.

Since the Fast block coder is a drop-in replacement for the J2K block coder, all of the above-mentioned JPIP functionalities are also available for interactive communication of media compressed using FBCOT. However, since Fast block bit-streams contain at most one Cleanup pass, followed at most by one SigProp and one MagRef coding pass, very little quality scalability is available. Quality scalability is important for responsive interactive communication via JPIP in bandwidth constrained environments, since it allows a server to stream content in quality progressive fashion and to redistribute bits within the WOI.

Fortunately, the limited quality scalability of FBCOT can be overcome by transcoding. In some applications, imagery or video may be compressed using FBCOT for maximum throughput, but transcoded on demand to J2K block bit-streams by a JPIP server, for efficient and responsive communication. J2K content that is communicated via JPIP may then be transcoded at client side to the Fast representation, whenever a code-block is first decoded, so that subsequent decodes consume much less energy/time[1].

TRANSCODING APPLICATIONS AND PERFORMANCE

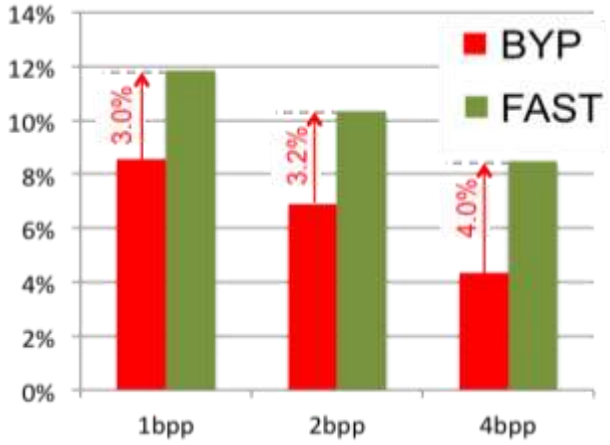

Figure 4 -- relative size increase on transcode

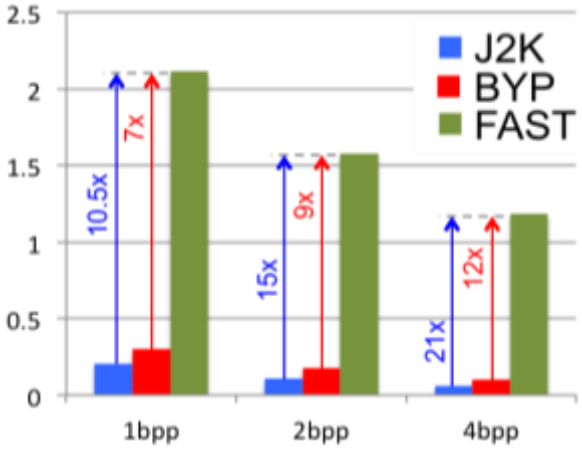

Figure 5 -- decode throughput, Gsamples/s

We provide evidence here for the compression efficiency and high throughput of the Fast block coding algorithm in a transcoding context. Original J2K block bit-streams are compressed to 1, 2 and 4 bits/pel (bpp), then transcoded to the FBCOT format. Figure 4 shows the average increase in compressed size associated with transcoding, with averages taken across most images from the original JPEG 2000 test corpus (Aerial2, Bike, Café, Cats, Cmp2, Finger, Goldhill, Hotel, Mat, Seismic, Texture1, Texture2, Tools, Ultrasound, Water, Woman and Xray), to which we add 2 high resolution colour images (Waltham1 and Waltham2) that were later contributed for evaluation of J2K. For reference, we also report the impact of transcoding to the “all-bypass” mode of the J2K block coder, denoted here as “BYP” – this is the fastest available mode offered by the J2K block coder, in which arithmetic coding is bypassed in all SigProp and MagRef passes.

[1] In an interactive JPIP browsing session, code-blocks tend to be decoded many times as a user pans, zooms and otherwise navigates through large media – it is convenient and much less resource intensive to cache coded content rather than rendered samples.

Figure 5 presents average decoding throughputs for J2K, BYP and Fast block coding algorithms. These results are obtained by measuring throughput of just the block decoder, on a late-2016 15" Macbook Pro laptop equipped with a 2.6GHz Skylake CPU. The test is single-threaded, but the measured time spent processing each code-block is divided by 4 to account for the fact that the CPU has 4 cores. Results are averaged over the same test images mentioned above. We note that the transform and quantization steps, not included in the throughput calculations, can be performed in around 1.5 clocks per sample on a single core of the CPU used here, which can at least partially be offset by hyperthreading within an i7 CPU. The overall speedup relative to J2K is 10.5x, 15x and 21x, at 1, 2 and 4 bits/pel, respectively. Most of the test images are greyscale, for which the Gsample/s measure is equivalent to Gpel/s. For the 5 RGB test images in the corpus, the average throughput is 2.4 Gpel/s, 1.5 Gpel/s and 0.88 Gpel/s at 1, 2 and 4 bits/pel, respectively.

All of these results are obtained using the well-known commercial Kakadu toolkit for JPEG 2000 (version 7.9.2), which is generally considered to be a very efficient implementation. Note, however, that a heavily optimized "Speed-Pack" version of Kakadu can accelerate J2K block encoding and decoding by around 1.5x.

HIGH PERFORMANCE END-TO-END VIDEO COMPRESSION WITH FBCOT

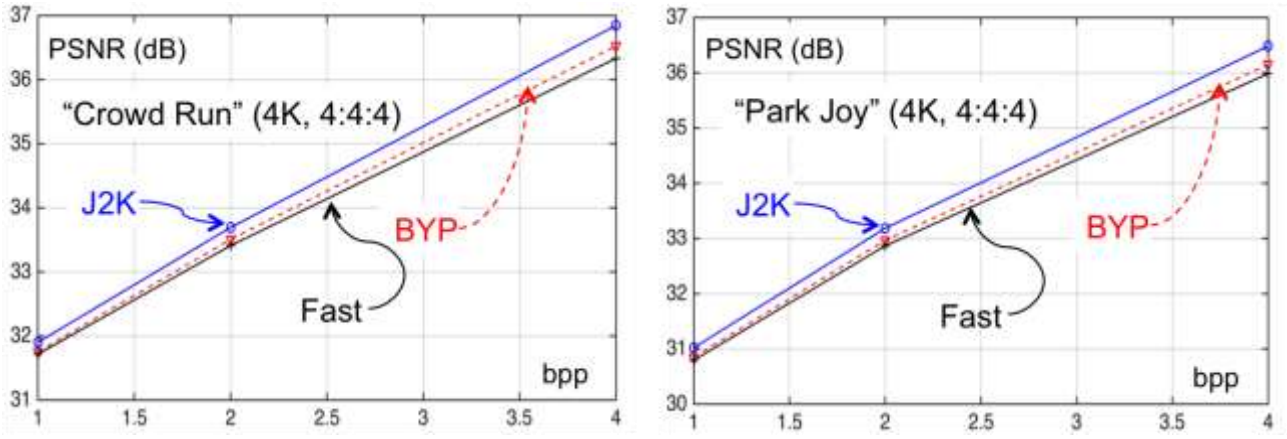

Figure 6 -- video compression efficiency of FBCOT vs. J2K and its BYP mode

Here we consider the throughput and efficiency of the FBCOT algorithm, including the Fast block coder and an efficient scheme for dynamically estimating the set of coding passes to encode for each code-block, after which the PCRD-opt algorithm decides which coding passes to include in each frame's code-stream. We generate at most 6 coding passes per block (2 x Cleanup, SigProp and MagRef). The algorithm uses statistics collected compressing earlier frames to decide which passes are actually encoded. Figure 6 compares the average PSNR (from the average squared error over all channels and frames) for two well-known 4K 4:4:4 60Hz video sequences. The comparison involves regular J2K, the "all-bypass" variant (BYP) and FBCOT (Fast); results are obtained using 32x32 code-blocks (specified by the J2K IMF profiles) and bit-rates of 1, 2 and 4 bpp.

Table 1 provides a breakdown of the time spent processing each coding pass of the Fast block encoder and decoder during FBCOT compression and subsequent decompression of 4K 4:4:4 video, measured in clock cycles per sample (single threaded). To obtain these

results, a Linux platform is used to lock the clock of an i7 Skylake CPU to its nominal frequency of 3.4GHz (no turbo) and the timestamp counter is then used to measure clock cycles. The individual coding pass complexities are measured with respect to the number of samples processed by that coding pass, while the "Total" figures measure the total number of clock cycles for the entire video, divided by the number of samples in the video. The table shows results for both 32x32 and 64x64 code-blocks, since our current implementation is not fully optimized for the 32x32 case.

	32x32 code-blocks (clks/sample)				64x64 code-blocks (clks/sample)			
	Cleanup	SigProp	MagRef	**Total**	Cleanup	SigProp	MagRef	**Total**
1 bpp encode	*4.46*	1.51	0.66	**7.9**	*3.47*	0.76	0.29	**5.6**
2 bpp encode	*4.79*	1.58	0.73	**9.2**	*3.95*	0.82	0.33	**6.9**
1 bpp decode	8.86	3.83	0.98	**1.53**	7.33	3.1	0.8	**1.45**
2 bpp decode	8.92	3.84	0.96	**2.95**	7.58	3.28	0.81	**2.81**

Table 1 -- Coding pass and overall complexity on Skylake in clocks/sample.

Encoding totals in the table are larger than the sum of the individual coding pass clocks/sample, because up to 6 coding passes are encoded for each code-block. This is done to achieve tight rate control via PCRD-opt, so that each frame has a fixed compressed size. If tight rate control is not required, a quantizer-driven leaky-bucket rate control approach may be used, as in typical video codecs and with JPEG compression. Then only one Cleanup pass need be performed, yielding extremely high throughputs (see *red italic* entries). Decoding totals in the table are smaller than the sum of the individual coding pass clocks/sample, because each code-block may use 0, 1, 2 or 3 coding passes.

SUMMARY AND FUTURE PROSPECTS

The Fast block coder and FBCOT compression framework presented here make it possible for a much larger set of applications and systems to benefit from the rich feature set of JPEG 2000. Reported CPU throughputs and efficiency already suggest that a future High Throughput J2K standard could be both faster and more efficient than JPEG-1 (ISO/IEC 10918-1). Also, reversible transcoding to J2K at the block level can avoid any sacrifice in performance, enabling highly efficient interactive streaming of image/video content, with arbitrary windows of interest. These features provide a strong argument for the incorporation of high throughput J2K solutions within cameras, servers, browsers and mobile devices, especially at high resolutions and for VR and 360 degree content.

REFERENCES

1. Taubman, D. and Marcellin, M. 2002. JPEG2000: Image compression fundamentals, standards and practice. Kluwer Academic Publishers.

2. Taubman, D. and Prandolini, R. 2003. Architecture, philosophy and performance of JPIP: internet protocol standard for JPEG 2000, Int. Symp. Visual Comm. and Image Proc. (VCIP), vol. 5150, pp. 649-663.

3. Taubman, D. 2000. High performance scalable image compression with EBCOT. IEEE Transactions on Image Processing, vol. 9, no. 8, pp. 12151-1170.