GPU-FRIENDLY EBCOT VARIANT WITH SINGLE-PASS SCAN ORDER AND RAW BIT PLANE CODING

Volker Bruns, Miguel Á. Martínez-del-Amor, Heiko Sparenberg

Fraunhofer Institute for Integrated Circuits, Erlangen, Germany

ABSTRACT

A major drawback of JPEG 2000 is the computational complexity of its entropy coder named Embedded Block Coder with Optimized Truncation (EBCOT). This paper proposes two alterations to the original algorithm that seek to improve the trade-off between compression efficiency and throughput. Firstly, magnitude bits within a bit plane are not prioritized by their significance anymore, but instead coded in a single pass instead of three, reducing the amount of expensive memory accesses at the cost of fewer truncation points. Secondly, low bit planes can entirely bypass the arithmetic coder and thus do not require any context-modelling. Both the encoder and decoder can process such bit planes in a sample-parallel fashion.

Experiments show average speed-ups of 1.6x (1.8x) for the encoder and 1.5x (1.9x) for the decoder, when beginning raw-coding after the fourth (third) bit plane, while the data rate increases only by 1.15x (1.3x).

Index Terms— JPEG 2000, GPGPU, EBCOT

1. INTRODUCTION

A popular field of use for JPEG 2000 [1] is in video post-production where it was chosen by the *Society of Motion Picture Technology Experts* (SMPTE) to be employed in both the *Digital Cinema Package* (DCP) format [2] and, more recently, in the *Interoperable Master Format* (IMF) [3]. The authors have developed an encoder and decoder that run on a GPU using NVIDIA's *CUDA* technology. The majority of the time is spent in EBCOT's context modelling and arithmetic coding routines. For DCPs, a maximum data rate of 250 Mbit/s is defined, for IMF packages it can be as high as 800 Mbit/s. In [4] it was shown that at high data rates the EBCOT compression efficiency is poor for the lower bit planes that contain mainly noise. A high throughput variant of EBCOT that can be used during post production and permits downstream lossless conversion to DCI or IMF profiles would be very beneficial to have. The standard addresses the high data rate scenario by defining the optional selective bypassing mode where part of the samples bypass the arithmetic coder and are coded raw. It only imposes a minor penalty, if any, on the compression efficiency. However, the speed-up achieved with that mode in a GPU implementation was shown to be only mediocre [4].

This paper proposes two new modes that decrease the complexity further and are especially useful to coders running on massively parallel architectures. The research questions that arise are: How much of a speed-up can be achieved? What is the cost in terms of diminished compression efficiency?

Sections 1 and 2 provide the required background about EBCOT, its latest variants and current GPU implementations. Sections 3 and 4 discuss the proposed single-pass and raw modes. Experimental results are presented in section 5 and summarized in section 6.

2. BACKGROUND

2.1. Review of EBCOT

Following the multi component transform, wavelet transform and quantization, subbands of quantized wavelet coefficients are split into non-overlapping code-blocks that are compressed independently into bit streams. Rate control is carried out by subsequently truncating these embedded bit streams in order to meet a global maximum data rate constraint. A single bit stream is formed by feeding the magnitude and sign-bits of each sample into a context-adaptive binary arithmetic coder (*MQ-coder*). The scan order goes first by bit plane, starting from the most significant magnitude bit plane that does not consist entirely of zero-bits, and then by stripe, where each stripe contains four rows. At any given bit plane the goal is to code samples that are likely to turn significant before bits that merely refine an already non-zero magnitude. This is achieved by scanning each bit plane in three passes. The *significance propagation pass* (SPP) codes all those samples that are still zero to the decoder's knowledge, but are predicted to turn significant as inferred from their 3x3-neighborhood. The *magnitude refinement pass* (MRP) codes samples that already turned significant in a previous bit plane. The *clean-up pass* (CUP) codes all remaining samples. A sample's sign-bit is coded immediately following its first significant magnitude bit. [5]

2.2 State of the Art

Other variants of EBCOT have previously been suggested in the literature. [6] proposes an "ultrafast" mode, where EBCOT is replaced with a fixed codebook Huffman coder preceded by an optional prediction step. A comparison of CPU implementations results in an achieved speed-up of 1.2-1.5x for the encoder and 2.2x-2.9x for the decoder. On the down side, the resulting codestream is no longer embedded and a post-compression rate control is not possible anymore.

EBCOT variants especially tailored for GPU architectures are described in [7] and [8]. [7] replaces the MQ-coder with a classical arithmetic coder and elaborates how to parallelize it. It also examines how the code-block size impacts the GPU occupancy and memory utilization.

In [8], a code-block is split into columns that are coded in parallel by multiple arithmetic coders that each produce fixed-length code words which can be interleaved in a deterministic way. To get around the problem, that each coder would have to initially adapt its probability estimates to the content at hand, a stationary pre-trained probability model is used instead. The compression efficiency has been evaluated and is reported to be equal to EBCOT. Throughput measurements are not yet available, but are expected to be very competitive.

2.3 EBCOT on a parallel Architecture

When targeting a massively parallel architecture such as a GPU, the goal is to break down the problem at hand into as many parallelizable work units as possible. For each bit plane, an encoder can model the samples' contexts in a sample-parallel fashion by using the iterative significance propagation algorithm presented in [9]. The resulting set of context-decision pairs is subsequently fed into the MQ-coder. Due to its context-adaptive nature, this step does not expose any intra-block parallelism. Therefore, an interleaved two-kernel structure presents itself. For each bit plane p, where $p=0$ is the LSB, the host launches a sample-parallel context-modelling kernel ($CM\text{-}kernel_p$) followed by a block-parallel arithmetic-coding-kernel ($MQ\text{-}kernel_p$).

The EBCOT decoding routine does not expose any intra-block parallelism since the significance propagation for a bit plane cannot be precomputed. Context-modelling and MQ-decoding are tightly coupled in a feedback loop: The context of any sample $X[n]_p$, where n is the location (x,y) with $0 \leq x < block\ width$ and $0 \leq y < block\ height$, cannot be computed without first decoding sample $X[n\text{-}1]_p$. A GPU implementation can therefore combine both tasks in a single kernel and need not launch it individually for each bit plane. Instead the kernel loops over all bit planes with three passes each.

On a modern GPU with thousands of cores the block-wise *MQ-kernel* tends to under-occupy the GPU due to its coarse parallelism. This effect worsens as the number of code-blocks with outstanding bit planes to be processed decreases with every processed bit plane. Implementations can counter-act by processing blocks from a group-of-pictures (GOP) simultaneously at the cost of an increased latency. The implementation at hand saturates at a GOP-size of 8 for 2K 4:4:4 images with approximately 5000 blocks per image.

3. SINGLE-PASS MODE

The *CM-* and *MQ-kernels* are memory bound, which means that the execution of instructions is often stalled by pending memory requests. Accordingly, the throughput can be increased by reducing required read- or write accesses to memory. The encoder's *CM-kernel* can be launched with a layout as finely grained as one thread per sample. After the initial iterative significance propagation, each thread determines their sample's context by examining the neighbors. Due to the three-pass-scan-order, the context-decision pairs need to be sorted by pass type before they can be handed over to the *MQ-kernel* via global memory. Each thread needs to find out at which offset to write their SPP-, MRP- or CUP-context-decision pairs. This can be achieved collaboratively by running three prefix-sum scans [10].

The proposed single-pass mode renders this additional burden unnecessary by giving up the separation into three passes altogether. The encoder's *CM-kernel* then requires only a single prefix-sum scan in order for each thread to determine where to write their context-decision pairs. Context-labeling is not affected by the new scan order and can remain unchanged. The decoder benefits as well in that it does then only need to scan through all samples once per bit plane. Additionally, the sample state representation is simplified by one bit as it is no longer necessary to remember for insignificant samples whether or not they had already been coded in the SPP in order to determine their membership to the CUP. The altered coding procedure is laid out in Algorithm 1.

There are two downsides to giving up the pass separation. Firstly, the number of eligible truncation points is decreased to a third, leaving the *Post-Compression-Rate-*

Algorithm 1 – single-pass mode

```
1.  foreach bit plane
2.    foreach sample in stripe-oriented scan-order
3.      if (sample is not significant and
4.          any neighbor is significant)
5.        zeroCoding()
6.        if (magnitude bit is 1)
7.          markSampleAsSignificant()
8.          signCoding()
9.      else if (sample is significant)
10.       magnitudeRefinementCoding()
11.     else if (first row in stripe and
12.       all stripe-column neighbors insignificant)
13.       runLengthCoding()
14.       jumpToEndOfRun()
15.     else
16.       zeroCoding()
17.       if (magnitude bit is 1)
18.         markSampleAsSignificant()
19.         signCoding()
```

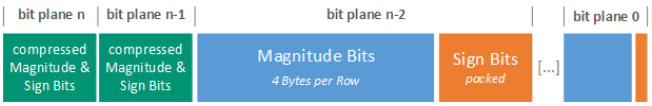

Figure 3 – Code stream structure when raw coding is entered after the 2nd significant bit plane. Code-block size: 32x32

Distortion-Optimizer (PCRD-Opt) less flexibility when the maximum data rate constraint is exceeded. It can be argued that this limitation is not severe when compressing to high data rates as is the case in the use-case examined in this paper. Research has shown that the coding performance achieved by various scanning order strategies is virtually identical at the end of each bit plane [11].

Secondly, the neighborhood taken into consideration when computing the context for any of the four types of coding operation needs to be constrained to the scan-order past. Whereas in the original mode, operations carried out during the MRP or CUP can already leverage the information whether or not any of the neighbors, past or future, has turned significant during the current bit plane's SPP, this is not possible anymore in the proposed variant. This might slightly impact the compression efficiency.

4. RAW CODING MODE

Part-1 of the standard defines the selective bypassing mode. When enabled, all symbols coded during the SP- and MR-passes starting from the fifth coded magnitude bit plane are not arithmetically coded anymore, but instead appended directly to the bit stream. The underlying assumption is that the compression efficiency for these two pass types is poor at low bit planes, while for the CUP it is still reasonable to invoke the MQ-coder. The bit stream is then terminated at every switch between the raw and coded mode. A parallel implementation can benefit from this mode by separating the raw coding phases into a separate kernel that can be executed in a sample-parallel fashion [2]. However, even the partially bypassed bit planes still necessitate the extra tasks of evaluating and updating each sample's state, including the significance propagation, in order to be able to collect all symbols and order them by pass type.

In line with the single-pass mode, the proposed raw-coding mode is also processed bit plane-wise without a pass separation. Instead, magnitude bits are directly concatenated into a bit stream. A block width of 32, which is used in DCI and IMF profiles, conveniently matches the SIMD group size and register size for NVIDIA CUDA devices. The encoder can collect the magnitude bits from a single row into a 32-bit register by using the warp ballot intrinsic and then write it out to memory. All rows of all blocks can be written in parallel. For a block size of 32x32 the magnitude bits amount to a total of 128 bytes per bit plane.

Since it is not guaranteed that every sample has already turned significant by the point the first bypassed bit plane is processed, not all sign-bits have necessarily been coded yet. With the goal of maintaining the embeddedness-property of

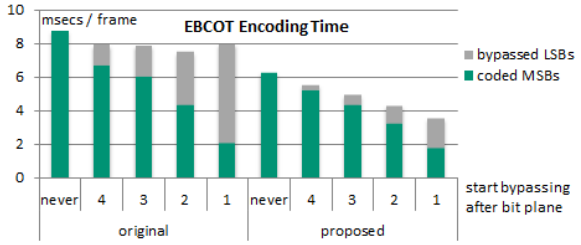

Figure 1 – source: "Tears of Steel", frames 1290-1547, 4096x1714

the created code-stream, a sample's sign-bit should be coded immediately following its first significant magnitude bit. However, due to the lack of sub-bit plane truncation points, magnitude and sign bits from one bit plane don't need to be interleaved. Instead sign-bits for newly significant samples are stored after the magnitude bits. This enables a GPU implementation to exploit that magnitude bits are laid out in a regular grid and can be accessed in parallel. Figure1 shows the resulting code stream structure.

The proposed implementation strategy is to first collect the sign-bits row-wise by again using the warp ballot intrinsic. A first ballot collects which positions in a row have turned significant; a second ballot collects the actual sign-bits. One thread per row then writes the rows' new sign-bits tightly packed to the bit stream. Since multiple threads might simultaneously need to write to the same byte, bits are stored using a bitwise atomic or-operation [12]. The offset beyond the end of the magnitude bits can be computed by scanning the prefix-sum across the numbers of sign-bits in each row.

5. EXPERIMENTAL RESULTS

Figure 2 compares the EBCOT encoding runtime of the original three-pass mode with and without selective arithmetic coding with the new proposed single-pass and raw-coding modes. All experiments have been carried out on an NVIDIA GeForce GTX 1080. The speed-up due solely to the proposed single-pass mode is approximately 1.4x and can be seen by comparing the two single-colored bars, where bit planes *never* bypass the arithmetic coder. The standard-compliant bypassing mode brings only little benefit for the throughput for a GPU implementation. The proposed raw coding of the lower bit planes, on the other hand, yields an additional speed-up that grows as more bit planes are bypassed.

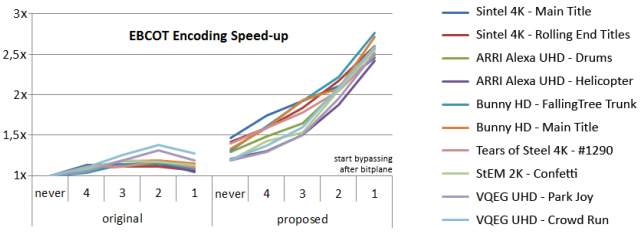

Figure 2 - EBCOT encoding speed-up relative to original three-pass mode without bypassing

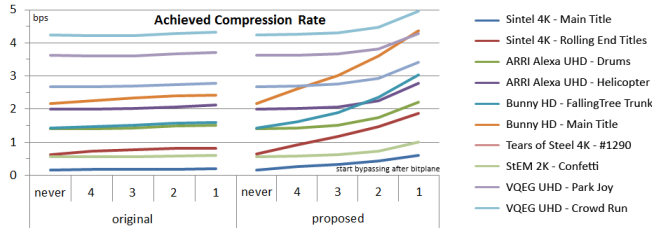

Figure 5 – Achieved compression rate measured in bits/sample (bps)

Figure 3 presents an overview of the achieved speed-ups for a range of high-resolution 48 bit RGB 4:4:4 image sequences. These sequences were chosen instead of the standard test still images, since processing blocks from multiple images in parallel increases the throughput significantly. Duplicating a still image would produce unrealistic results since then the execution divergence due to different image contents is not taken into account.

The gain is highest for images with a low-entropy since then the GPU is most heavily under-occupied without the proposed modes as only few code-blocks remain active in the lower bit planes. The average speed-up for the single-pass mode is 1.3x with a factor of 1.2x on the low end for the detail-rich *VQEG* or *StEM* sequences and almost 1.5x on the high end for the low entropy images from the *Sintel* or *Big Buck Bunny* sequences. When enabling the raw-coding mode after the fourth (third) significant bit plane, the average speed-up is increased to 1.6x (1.8x).

The impact of the single-pass mode on the decoder is lower, but still yields an average speed-up of 1.16x. The proposed raw-coding mode works better in the decoder, though, with speed-ups of 1.5x and 1.9x when enabling it after the fourth or third plane, respectively.

The down side to the proposed modes is a diminished compression efficiency. Figure 4 plots how the achieved compression rate decreases as more bit planes are subjected to raw coding. The compression rate increase due to the single-pass mode is only about 0.1%. As expected, it grows significantly as more bit planes are coded raw.

All sequences have been compressed with a sufficiently high maximum data rate so that no bit streams had to be truncated. It has yet to be examined how the quality compares when compressing to a fixed target rate, given that the single-pass mode yields fewer truncation points.

Ultimately, an assessment of whether the gain in the form of increased encoding or decoding throughput outweighs the cost in terms of an increased data rate depends on the use case. An attempt is made here by plotting the speed-up vs. the relative increase in data rate (Figure 5). Every curve connects five points, which represent the result for no bypassing, bypassing after 4, 3, 2 or 1 significant bit planes. For those points that lie below the diagonal dashed line, the speed-up is higher than the relative increase in data rate.

All except two sequences stay entirely below the dashed line. The two exceptions correspond to low entropy

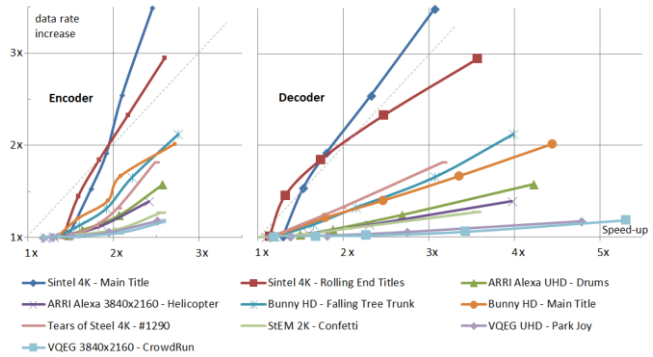

Figure 4 – Data rate increase vs. speed-up

sequences that had a high compression efficiency and throughput to begin with and could therefore be regarded as uncritical.

6. CONCLUSION

Based on an examination of the bottlenecks of a JPEG 2000 implementation for GPUs, we proposed two new modes for its entropy coder EBCOT that lead to a higher throughput. The variant is compatible with the JPEG 2000 framework and can be losslessly transcoded into standard-compliant profiles. The resulting codestream is still embedded so that decoders can apply the technique of code-pass skipping in order to increase their throughput further when it is not a strict requirement to fully decode the compressed images, e.g. when previewing 36 bit DCPs on a 24 bit computer monitor [13].

The single-pass mode alone yields speed-ups between 1.2x and 1.5x for the encoder and 1.05x to 1.3x for the decoder at virtually no increase in compression rate.

The throughput can be further increased by enabling the raw-coding mode on top of the single-pass mode. In contrast to the selective-arithmetic coding bypass mode defined in the JPEG 2000 standard, this proposed mode can be leveraged by a GPU implementation to increase the throughput significantly. This comes at the cost of an increased data rate, but the relative speed-up outweighs the relative increase in data rate for all tested detail-rich sequences. It presents an opportunity to flexibly configure the trade-off between speed and compression rate.

7. REFERENCES

[1] M. Boliek (Ed.), ”Information Technology - The JPEG2000 image coding system: Part 1”, ISO/IEC 15444-1, 2000

[2] A. Bilgin, M.W. Marcelling, “JPEG2000 for Digital Cinema”, *IEEE International Symposium on Circuits and Systems*, pp. 3878-3881, May 2006

[3] SMPTE ST 2067-21:2014. Interoperable Master Format – Application #2 Extended

[4] V. Bruns and M. A. Martínez-del-Amor, "Sample-Parallel Execution of EBCOT in Fast Mode", *32nd Picture Coding Symposium*, Nuremberg, Erlangen, Dec. 2016

[5] D. Taubman, "High performance scalable image compression with EBCOT", *IEEE Transactions on Image Processing, Vol. 9 No. 7*, pp. 1151-1170, 2000

[6] T. Richter and S. Simon, "On the JPEG 2000 ultrafast mode", *19th IEEE International Conference on Image Processing*, Oct 2012

[7] R. Le, I.R. Bahar and J.L. Mundy, "A novel parallel Tier-1 coder for JPEG2000 using GPUs", *IEEE 9th Symposium on Application Specific Processors*, pp. 129-136, Jun 2011

[8] F. Aulí-Llinàs, P. Enfedaque, J.C. Moure and V. Sanchez, Bitplane Image Coding With Parallel Coefficient Processing, *IEEE Transactions on Image Processing, Vol. 25, No. 1*, Jan 2016

[9] J. Matela, V. Rusnak and P. Holub, "GPU-Based Sample-Parallel Context Modelling for EBCOT in JPEG2000," *Sixth Doctoral Workshop on Math. and Eng. Methods in Computer Science (MEMICS'10)*, pp. 77-84, 2010

[10] G.E. Blelloch, "Scans as Primitive Parallel Operations", *IEEE Transactions on Computers, Vol. 38, No*. 11, pp. 1526-1538, Nov 1989

[11] F. Aulí-Llinàs and M.W. Marcellin, "Scanning Order Strategies for Bitplane Image Coding", *IEEE Transaction on Image Processing, Vol. 21, No. 4*, April 2012

[12] A. Balevic, "Parallel variable-length encoding on GPGPUs," *Proceedings of the 2009 international conference on Parallel Processing*, EuroPar'09, Berlin, pp. 26-35, 2010

[13] V. Bruns and H. Sparenberg, "Comparison of Code-Pass-Skipping Strategies for Accelerating a JPEG 2000 Decoder", 15. *ITG-Fachtagung für Elektronische Medien*, Dortmund, Feb. 2013