THREE-LEVEL PARALLEL HIGH SPEED ARCHITECTURE FOR EBCOT IN JPEG2000

Yijun Li and Magdy Bayoumi

The Center for Advanced Computer Studies
University of Louisiana at Lafayette, Lafayette, LA, 70504, USA
{yxl4444,mab}@cacs.louisiana.edu

ABSTRACT

In this paper, a multi-level parallel high speed architecture for Embedded Block Coding with Optimized Truncation (EBCOT) tier-1 in JPEG2000 is proposed. To increase the system throughput, this architecture adopts three levels of parallelism: 1. the parallelism among bit-planes: all the bit-planes can be processed simultaneously; 2. the parallelism among three pass scannings: three passes scan one bit-plane in parallel; 3. the parallelism among coding bits: bits that are coded in different passes can be coded simultaneously without any conflict. The experiment results show that the proposed architecture can encode one code block with size $N \times N$ in only $0.6 \times N \times N$ clock cycles and is twice as fast as the fastest architecture in the literatures so far.

1. INTRODUCTION

In January 2001, JPEG2000 was introduced by ISO/IEC JTC1/SC20/WG1 as a new image compression standard[1]. This new standard supports the rich set of features that are not available in existing JPEG standard, such as excellent low bit-rate performance, both lossy and lossless encoding in one algorithm, random codestream access, precise single-pass rate control, region-of-interest coding and improved error resiliency.

The block diagram of JPEG2000 is shown in Fig. 1. The original image data is divided into non-overlapping rectangular tiles. Then either (5,3) discrete wavelet transform (DWT) supporting loss-less compression or (9,7) DWT supporting lossy compression are performed on the tiles by filtering each row and column of the image tiles with a high-pass and low-pass filter. Filtering the image in the DWT phase creates a set of DWT subbands (LL, HL, LH, HH). If lossy compression is chosen, the wavelet coefficients in DWT subbands are scalar-quantized. Each wavelet subband is divided into code blocks. Then the wavelet coefficients in code blocks are entropy coded by using EBCOT algorithm.

Thanks to the support of the U.S. Department of Energy (DoE), EETAPP program DE97ER12220, the Governor's Information Technology Initiative, and the support of NSF, INF 6-001-006.

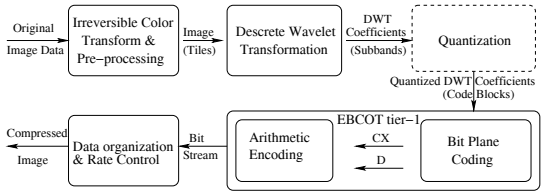

Fig. 1. Block Diagram of JPEG2000

Finally, data ordering and rate control organize the compressed data into a feature-rich codestream, i.e. the compressed image.

As entropy coding, EBCOT algorithm is complicated and full with bit operations that cannot be implemented efficiently in software. Profiling techniques in [2] show the EBCOT algorithm is a huge time-consuming part (typically more than 50%). As results, an unefficient EBCOT implementation becomes a bottleneck for tremendous data throughput that are often required by multimedia systems, especially, real-time applications. Since bit operations are more suitable to hardware implementation and parallelism usually can increase the system throughput, multi-level parallel EBCOT hardware implementation should be a solution for the bottleneck mentioned above.

2. EBCOT ALGORITHM IN JPEG2000

As shown in Fig. 1, EBCOT algorithm is divided into two phases: bit-plane coding phase and adaptive arithmetic encoding (AE) phase. Bit-plane coding phase constructs context information for each bit, i.e. context (CX) and decision (D). By using the context (CX), AE adaptively encodes the decision (D) bit by bit to achieve high coding efficiency.

2.1. Bit-Plane Coding

The DWT coefficients in a code-block memory are stored as binary values. Each binary value can be seen as a sequence of bits with positions from n to 1 if the binary value has n bits. All the bits at the same position composes a bit-plane. For example, all the Least Significant Bits (LSBs) of the DWT coefficients in a code-block compose a bit-plane

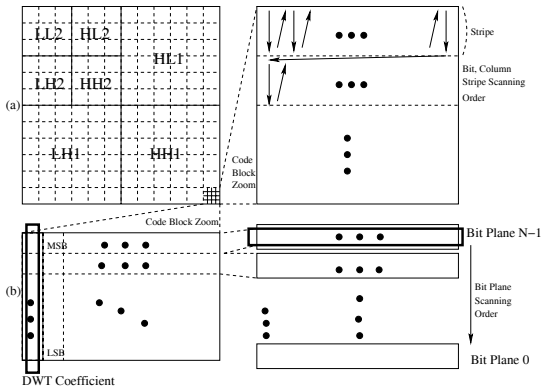

Fig. 2. EBCOT coding order

because they are on the same position 1. Each bit plane is divided into stripes that are continuous four rows of a bit-plane, i.e. a bit-plane with size $N \times N$ should have $N/4$ stripes. So one column of a stripe has 4 bits. Bit plane coding phase contains three passes: Significant Pass (Pass 1), Magnitude Refinement Pass (Pass 2) and Clean-up Pass (Pass 3). Each pass should scan a bit-plane in order: bit by bit, column by column and stripe by stripe. Then a code block is scanned bit-plane by bit-plane. Fig. 2 provides a topview (Fig. 2 (a)) and a sideview (Fig. 2 (b)) for a code block and scanning order of passes.

A DWT coefficient is associated with a significance state. Before starting the three passes, all the significance states of a code-block are initialized as insignificant. During pass scanning, the significance state of a wavelet coefficient is changed from insignificant state into significant state whenever Most Significant Bit (MSB) of the wavelet coefficient is coded. So all the bits of a DWT coefficient before MSB coding is considered as insignificant for all the three passes and all the bits of a DWT coefficient after MSB coding is considered as significant for all the three passes. The significance states of one bit and its 8 neighbors determine which pass a bit is coded in and both CX and D. If one bit's significance state is insignificant, the bit is insignificant. Otherwise, the bit is significant. If a bit is insignificant and any neighbor is significant, the bit is coded in Pass 1. If a bit is significant, the bit is coded in Pass 2. The rest of all the bits are coded in Pass 3. During the scanning of the three passes, any bit can only be coded in one of three passes (Pass 1, Pass 2 or Pass 3).

2.2. AE termination

Since AE is a strictly serial process, AE termination is used to remove the dependency of two bit-streams. By using various AE termination, EBCOT coding can be processed in either serial mode or parallel mode. If coding in the serial mode, three passes scan a bit-plane in serial. After all the bit planes are scanned, AE terminates the bit stream. If coding in the parallel mode, three passes scan a code-block as same as in the serial mode. But AE terminates the bit stream at the end of each pass. This termination pattern removes the dependency between passes. As results, parallel mode may allow three passes scan a bit-plane simultaneously, since the AE probability interval of any pass doesn't depend on each other. Besides, to remove the dependency of the current stripe coding on the next stripe, stripe-causal mode is adopted, i.e. the neighbor contributions from the next stripe are always considered as 0. The parallel mode is more error-resilient, although its performance is slightly poorer than the serial mode (average degration PSNR is only 0.19dB)[3, 5].

3. RELATED WORK

By mapping either serial mode or parallel mode of EBCOT algorithm (the algorithm is described in Section 2), EBCOT architecture can be implemented in either serial mode or parallel mode. A serial mode architecture was introduced in [2], where three passes scan a bit-plane in serial. In this architecture, fetching operation is column-based, i.e. in a clock cycle, one column (4 bits) is fetched from memory instead of one bit to reduce the number of memory access. Pixel-Skipping techniques were adopted to skip the unnecessary bit evaluation. But Pixel-Skipping techniques cannot remove the unnecessary bit evaluation completely. So two parallel mode architectures were proposed in [3, 4], where three passes scan a bit-plane in parallel and column-based fetching operation was adopted as same as [2]. Moreover, the parallelism between 4 coding bits was introduced in [4], where two bits from different passes may be coded in the same clock cycle. In all the architectures above, all the bit-planes are coded in serial. So a memory is needed to associate the current significance states with a bit-plane. The parallel mode architectue that doesn't need the memory for significance states was proposed in [5], where 4 bits of a column are coded bit by bit and only one bit is coded in one clock cycle.

4. THE PROPOSED ARCHITECTURE

To achieve high system throughput, three levels of parallelism are adopted: 1. the parallelism among bit-planes: all the bit-planes can be processed simultaneously and significance state memory can be removed by predicting the significance state of each bit; 2. the parallelism among pass scannings: three passes scan one bit-plane in parallel, bits are coded immediately after they are evaluated and no processing time is wasted; 3. the parallelism among coding bits: When three bits that need to be coded in Pass 1, Pass 2 and Pass 3, respectively, they can be coded in the same clock cycle simultaneously by adding one extra primitive encoder for Pass 3.

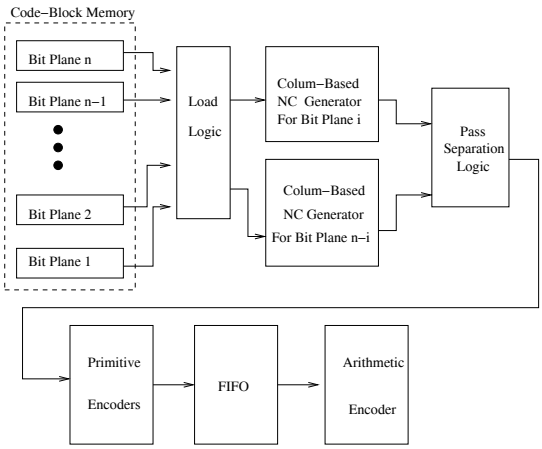

Fig. 3. The Proposed EBCOT Architecture

The proposed architecture is shown in Fig. 3. Code-block memory contains all the DWT coefficients of a code-block. Load logic model fetches 4 wavelet coefficients in one clock cycle from the code block memory. MSB of these DWT coefficients are used by load-logic model to initialize the significance states of bits. For each DWT coefficient, its bits above its MSB, including MSB, are initialized as insignificant and the others are initialized as significant. The initialized significance states are feeded into Column-based NC (Neighbor Contributions) generator models for evaluation of neighbor contributions. Note that these initial significance states are not the final values that can be used to calculate the neighbor contributions. More steps are needed to predict the final values. (Section 4.1 will talk about it in details).

The Column-based NC generator for bit-plane i is associated with a column of the bit-plane i. So n Column-based NC generators may code all the bit-planes simultaneously. Here, the bit-plane $n-i$ coding and the bit-plane i coding are combined together in the pass seperation logic model. (for simplicity, only bit-plane $n-i$ coding and bit-plane i coding are shown in Fig. 3). Two reasons motivate us to adopt this scheme. First, the number of context labels from the bit plane $n-i$ are usually less than the number of context labels from the bit plane i since the bit plane $n-i$ may have more bits coded by Run-Length coding. Run-Length coding may encode more than 1 bits but it generates only one context label. This combination will benefit FIFO (No big variance for data in FIFO). Second, it favors the parallelism among coding bits when 8 bits are coded together.

As shown in Fig. 4, each Column-based NC generator can evaluate neighbor contributions for 4 bits simultaneously. All the neighbor contributions are used to determine which pass a bit is belong to and what the context label is. In primitive encoder model, there are 2 encoders for Pass 1, 2 encoders for Pass 2 and 2 encoders for Pass 3. So the primitive encoder model may concurrently encode 2 bits in

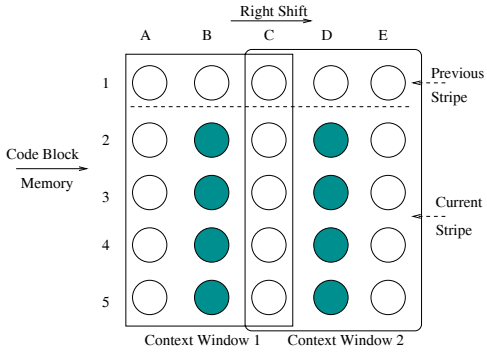

Fig. 4. Column-based DHV Generator

Pass 1, 2 bits in Pass 2 and 2 bits in Pass 3. Note that in our architecture run-length coder and sign-coder work in parallel with the primitive coder above. As results, column-based NC generators and primitive encoders provide the parallelism among passes and among bits. The primitive encoders write their outputs (CX and D) into FIFO. CX and D in FIFO are consumed by the pipelined high speed AE.

4.1. Removing data dependency

To support the multi-level parallelism, removing data dependency is essential. The data dependency among three passes can be removed if parallel mode of EBCOT is adopted. The data dependency among bit-planes and among coding bits can be removed if significance states can be predicted before three pass scannings. The change of significance states can occur in either Pass 1 or Pass 3. As results, Pass 1 depends on itself, Pass 2 depends on Pass 1 and Pass 3 depends on Pass 1 and itself.

To remove these dependency, two context windows shown in Fig. 4 were adopted, where one column is mapped to the fourth bit from the previous stripe and four bits from the current stripe. Column A, B and C forms the context window 1. Column C, D and E forms the context window 2. These two context windows are implemented as 5×5 register array. Context window 1 is used to indicate the Pass 1 coding state, i.e. whether the bits are coded in Pass 1 or not. It can be predicted by using initial significance states and its bit-value. One bit is coded in Pass 1 if it is insignificant and any neighbor is significant. After one column finishes coding, all values are right shifted by one column.

The Pass 1 coding states are shifted to context window 2. In context window 2, all the neighbor contributions and pass coding states (i.e. which pass the bit is belong to) can be calculated by using initial significance states from load logic model, the bit values and Pass 1 coding states. Pass 1 coding states indicate the bits coded in Pass 1. The bit is coded in Pass 2 if the bit is initialized as significant. The rest of all the bits are coded in Pass 3. The neighbor contribution calculation algorithm is shown as follows, where A, B, C, D, E, 1, 2, 3, 4, 5 are used to identify the location of bits as

shown in Fig. 4.

1. Calculate the significance states after Pass 1 scanning or Pass 3 scanning.
   (a) For C_1, D_1, E_1,
   $State_{afterPass1} = State_{initial} + Value$
   For other bits,
      i. if it is coded in Pass 1,
      $State_{afterPass1} = State_{initial} + Value$
      ii. else $State_{afterPass1} = State_{initial}$
   (b) For D_2, D_3, D_4, D_5 and E_2, E_3, E_4, E_5
      i. if it is coded in Pass 3,
      $State_{afterPass3} = State_{initial} + Value$
      ii. else $State_{afterPass3} = State_{afterPass1}$
2. Calculate the neighbor contributions
   (a) If D_i is coded in Pass 1
      i. (C_{i-1},C_i,C_{i+1},D_{i+1}), $State_{initial}$.
      ii. (D_{i-1},E_{i-1},E_i,E_{i+1}), $State_{afterPass1}$
   (b) If D_i is coded in Pass 2, all the neighbors should be $State_{afterPass1}$
   (c) If D_i is coded in Pass 3
      i. (C_{i-1},C_i,C_{i+1},D_{i+1}), $State_{afterPass1}$.
      ii. (D_{i-1},E_{i-1},E_i,E_{i+1}), $State_{afterPass3}$

5. EXPERIMENTAL RESULTS

To evaluate the proposed architecture, three-level parallelism is added to the EBCOT part of JasPer software[1] and the algorithm is also verified by Verilog HDL implementation. Some standard test images (grayscale, code-block size is 64×64) are used to estimate the clock cycles for encoding one code block. The images "cats" and "water" are from the JasPer software. The estimated clock cycles for three standard test images (lenna, cats and water) are shown in Table 1. The processing time comparison with related work is shown in Table 2, where n is the number of bit-planes that need to be coded in one code-block. Experimental results show that the proposed architecture is twice as fast as the high speed architecture [5] that is the fastest in literatures and can encode one code-block with size $N \times N$ in only $0.6 \times N \times N$ clock cycles.

Table 1. Coding One Code-Block (64×64)

Image File	lena	cats	water
Clock Cycles	2318	2386	2257

Compared with [5], the extra hardware (additional primitive encoders and pass-separation logic) are added to datapath for the multi-level parallelism. To evaluate this overhead, the proposed architecture is synthesized by using AMS $0.35\mu m$ CMOS technology. The data-path gate count and memory requirement for the proposed architecture and the gate count of the overhead are listed in Table 3.

Table 2. Processing Time for a $N \times N$ code-block

Architecture	[2]	[3]	[5]	Proposed
Time ($\times N^2$cycles)	$1.3 \times n$	n	1.2	0.6

Table 3. Architecture Implementation ($n = 10, N = 64$)

Memory(bits)	bit-plane	AE	Overhead
6228	43125	57680	9800

6. CONCLUSION

In this paper, a three-level parallel high speed architecture for EBCOT in JPEG2000 is proposed. The experiment results show the proposed architecture is twice as fast as the high speed architecture [5] that is the fastest architecture in the literatures so far.

7. REFERENCES

[1] M. Boliek, C. Christopoulos, and E. Majani (Editors), *JPEG2000 Part I Final Publication Draft*, ISO/IEC JTC1/SC29/WG1 N2678, July 2002.

[2] Chung-Jr Lian, Kuan-Fu Chen, Hong-Hui Chen, and Liang-Gee Chen, "Analysis and architecture design of block-coding engine for EBCOT in JPEG 2000," *IEEE Transactions on circuits and systems for video technology*, vol. 13, no. 3, pp. 219–230, March 2003.

[3] Jen-Shiun Chiang, Yu-Sen Lin, and Chang-Yo Hsieh, "Efficient pass-parallel architecture for EBCOT in JPEG2000," in *IEEE International Symposium on Circuits and Systems, 2002*, vol. 1, pp. 773–776, May 2002.

[4] Yijun Li, Ramy E.Aly, Magdy A.Bayoumi, and Samia A.Mashali, "Parallel high-speed architecture for EBCOT in JPEG2000," in *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, pp. 481–484, April 2003.

[5] Hung-Chi Fang, Tu-Chih Wang, Chung-Jr Lian, Te-Hao Chang, and Liang-Gee Chen, "High speed memory efficient EBCOT architecture for JPEG2000," in *IEEE International Symposium on Circuits and Systems (ISCAS 2003)*, vol. 2, pp. 736–739, May 2003.

[6] Michael D.Adams, *JasPer Software Reference Manual (Version 1.700.0)*, ISO/IEC JTC1/SC29/WG1 N2415, Febrary 2003.

[1] The JasPer software [6] is an official reference implementation of the JPEG-2000 Part-1 codec and has been included in the JPEG-2000 Part-5 standard (i.e., ISO/IEC 15444-5).