PARALLEL HIGH-SPEED ARCHITECTURE FOR EBCOT IN JPEG2000

Yijun Li, Ramy E.Aly,and Magdy A.Bayoumi

University of Louisiana at Lafayette
P.O.Box 44330
Lafayette, Louisiana 70504-4330, USA

Samia A. Mashali

Electronic Research Institute (ERI)
Egypt

ABSTRACT

In this paper, a parallel high-speed architecture for EBCOT is proposed, which based on the parallel mode in JPEG2000. It discovers the parallelism among three passes in bit-plane coding: two passes can work on coding operation without the addition of coding processing elements (PEs) by using parallel context modeling. So it manages to make two bits encoded in one clock cycle. In order to keep high throughput and reduce memory requirement, the pipelined pass-switching arithmetic encoder is adopted. The experimental results show that the proposed architecture reduces the processing time by more than 16.6% compared with the pass-parallel mode architecture in [3] and by more than 38% compared with serial mode architecture in [2].

1. INTRODUCTION

Recently, JPEG2000 has been introduced by ISO/IEC as new image compression standard, which provides a rich set of features that are not available in existing standards, such as pixel accuracy and resolution, random codestream access and region-of-interest coding.

The first step in JPEG2000, shown in Fig.1, is to divide the image into non-overlapping rectangular tiles. Then either (5,3) wavelet transform supporting loss-less compression or (9,7) wavelet transform supporting lossy compression are applied to the tile components. If lossy compression is chosen, the wavelet coefficients are scalar quantized. Then, each wavelet subband is divided into code blocks. The wavelet coefficients in code blocks are entropy coded by the embedded block coding with optimized truncation (EBCOT). Data ordering organizes the compressed data into a feature-rich code stream. In JPEG2000, EBCOT algorithm, as entropy coding, is an important and complicated component. It does bit-plane coding by using 3 passes. The context labels are obtained by using four primitive encoders. These labels determine how arithmetic encoder does and the data are encoded with arithmetic encoder. So the EBCOT entropy coder consumes too much time in JPEG2000 (typically more than 50%) [2].

In this paper, a parallel high-speed architecture for EBCOT is proposed, which supports the implementation of two encoded bits in one single cycle. In section 2, entropy coding algorithm (EBCOT) is discussed. In section 3, the parallel high-speed architecture is proposed. The main architecture components are explained in details in section 4. The simulation results are illustrated in section 5, and our conclusion is presented in the section 6.

2. ENTROPY CODING ALGORITHM (EBCOT)

2.1 Context Formation [1]

In terms of sign-magnitude representation, the quantized DWT coefficients in each code block consist of one sign bit-plane and several magnitude bit-planes. EBCOT codes these coefficients bit-plane by bit-plane, starting from the most significant bit-plane with at least a non-zero element to the least significant bit-plane. Each bit-plane is coded in three coding passes. Each bit in a bit-plane is coded once in only one of three coding passes. Every four rows in each bit-plane are called "stripe". In each pass, the bits are scanned stripe by stripe from top to down. Within a stripe, each 4-bits column is scanned column by column from left to right. Within a column, each bit location is scanned bit by bit from top to down.

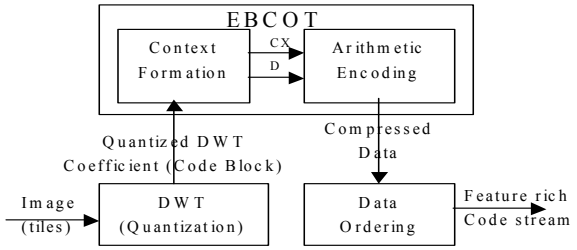

Fig.1 Block diagram of JPEG2000 Encoder

EBCOT has four primitives coding PEs: zero coding (ZC), sign coding (SC), magnitude refinement (MR), and run-length coding (RLC). Note that each bit location in a code-block is associated with a binary state variable called "significance state." Significance states are initialized to '0' (denotes insignificant). If the bit is insignificant and its value is '1', its significance state may become '1' (denotes significant). If the bit is insignificant and any one of its neighbors is significant, it is coded in significance propagation pass (pass 1) that is based on ZC primitive. If the bit is significant, it is coded in magnitude refinement pass (pass 2) that is based on MR primitive. Other bits are coded in cleanup pass (pass 3) that is based on ZC and RLC primitive. The coding primitives generate context labels based on the sign and significance status of 8-connect neighbors

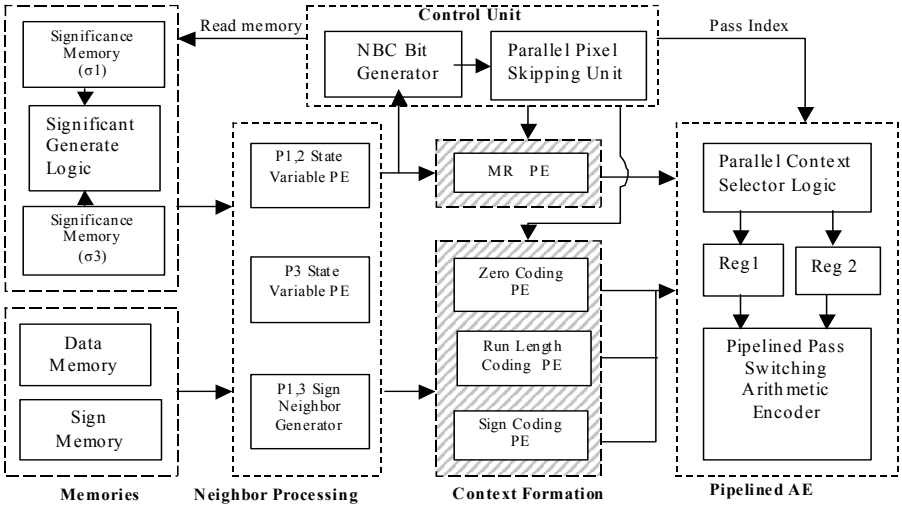

Fig.2 The proposed Architecture for EBCOT

of the current bit.

2.2 Adaptive Binary Arithmetic Coding[1]

After context labels are formed, they are provided into the adaptive binary arithmetic encoder (MQ coder), which is based on the recursive probability interval subdivision of Elias coding. For every context label, there is a corresponding state machine associated with it. The state machine is same for every label, but different label may work in the different states. It provides the probability estimation for MQ coder. Each current probability interval is partitioned into two subintervals: the more probable symbol (MPS) subinterval and the less probable symbol (LPS) subinterval. The probability estimations obtained from the state machine determine the values of code register C and interval register A for the next state. In order to keep the precision range required for register A, renormalization procedure (both of A and c are shifted) is adopted. It has a byte of compressed data removed and outputted from the high order bits of the code register C.

3.THE PROPOSED ARCHITECTURE

Column-based operation [2] is adopted in this proposed architecture. State variables read one column at a time from memories. 4 bits in each column are coded in three passes, but each bit is coded once in only one of three passes. MR PE is only needed in Pass 2, Zero Coding PE and RunLength Coding PE are only needed Pass 1 or Pass 3 and SignCoding PE is only needed when signs is encoded. The encoding of a bit belong to pass 1 or pass 3 don't conflict with the encoding of a bit in the same column belong to pass 2. Also, there is no conflict between pass 2 and pass 3, because they are in different context window logic and use different encoding processing elements (PE). Using this independent relation between encoded bits and by adding a simple logic control circuit, pass 2 can work in parallel with pass 1 or pass 3, shown in shade part in Fig.2. Pixel skipping [2] is also applied to skip the bit that is not coded in the current pass. However, the parallel working mode between pass 1, pass 3 and pass 2 seems different. In pass 2, the bit need to be skipped that is not coded in pass 2. In pass 1 or pass 3, the bit need to be skipped that is coded in pass 2 and distinguished for pass 1 and pass 3. Parallel pixel skipping unit, discussed in section 4.2, is adopted for solving this dependent check. In the result, at most two bits in each column can be coded in one single cycle.

Due to that two context labels can be done in one single cycle, two registers are adopted. One is for context label from pass 2, and other is for context label from pass 1 or pass 3 with a special particular bit to identify pass 1 or pass 3. One arithmetic encoder unit with pass-switching logic [3] is used for MQ coder. Moreover, the arithmetic encoder, discussed in section 4.3, is pipelined in two stages. So it uses less hardware instead of two or more arithmetic encoder units and removes the memory requirement for context label outputs. In order to solve the issues of dependence between pass 1, pass 2 and pass 3, parallel context window model, discussed in section 4.1, is adopted.

In the architecture, shown in Fig.2, the value and sign of DWT coefficients are stored into data and sign memory respectively. Since the significance state can only be changed in pass 1 or pass 3, there are two memories about significance states: significance state in significance memory ($\delta 1$) is associated with the bit location in pass 1 and significance state in significance memory ($\delta 3$) is associated with the bit location in pass 3. By using significant generator logic, the final significance states are supplied into the state variables. P1,2 state variable PE generates the neighbor attributions (D,H,V [1]) for pass 1 and pass 2 by using context window modeling, P3 state variable PE generates the neighbor attributions D,H,V for pass 3 by using context window modeling, and P1,3 sign neighbor generator generates sign neighbor contributions for pass 1 and pass 3. According to the neighbor contributions, need-be-coded (NBC) bits for pass 1

or pass 2 or pass 3 are determined respectively. The parallel pixel skipping unit takes these NBC as the input and generate the NBC index for pass 1, pass 2 and pass 3, i.e. what location bits are coded and whether or not they are coded in parallel. These NBC index will be as control signal to select neighbor contributions in the corresponding location, which are provided into the four primitive coding PE to generate the context labels. Due to the parallel pass processing, at most two context labels are generated in the single cycle. These output context labels are stored into two registers. Then the AEs catch the corresponding context labels and pass index from these two registers and do its arithmetic encoding by using pipelined pass switching arithmetic encoder.

4 ARCHITECTURE COMPONENTS

4.1Parallel Context Window Modeling

In order to get less memory access and more concurrency, the pixel skipping and the column-based operation are adopted [2]. Therefore, data are supplied to context window one column at a time. The 4 bits in a column can be coded in pass 1 or pass 2 or pass 3, but one bit is code once in only one of three passes. In pass 2, non-need-be-coded pixels in pass 2 are skipped. In pass 1 and pass 3, non-need-be-coded pixels in pass 1 and pass 3 are skipped too. Due to MR coding PE is independent from Zero coding PE and RLC coding PE. So they can do their coding in parallel. The two bits, one of which is from pass 2 and the other from pass 1 or pass 3, are then coded by four coding primitives. So at most two bits can be coded in a single cycle.

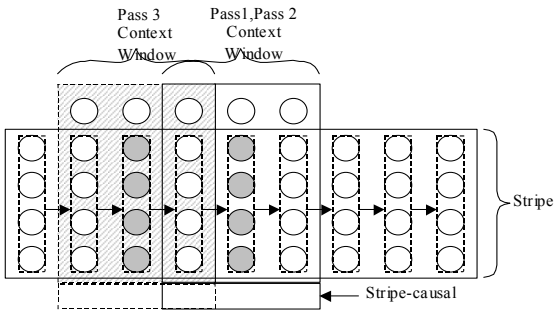

Fig.3 Context Window Logic

Further, the dependent relationship exists between the three coding passes in the parallel coding mode. The pass 3 context window depends on the change in the significant status of the bit in the processed column, which is occurred during pass 1,pass 2 context window. This means pass 3 context window has to be at least 2 columns apart from pass 1, pass 2 context window, to give a chance for pass 1,pass 2 context window to change the significant status of the bits in the processed column, if needed. In the proposed architecture, we use the context window logic, shown in Fig.3, in which pass 3 context window lags 2 columns behind pass 1, pass 2 context window to eliminate the dependence of pass 3 on pass 1. “Stripe-causal” mode [7] is also adopted to eliminate the dependence of coding operation on the next stripe. Moreover, in order to reduce the memory requirement, only two significance state variables, δ1 and δ3, are introduced [3]. δ1 and δ3 represent the significant status in pass 1 and pass 3, respectively. So when first MR coding is applied for the bit, its significance state can be represent in the term of δ1 XOR δ3. For the bit in pass 1 and pass 3, its significance states are equal to δ1 or δ3.

4.2 Parallel Pixel Skipping Unit

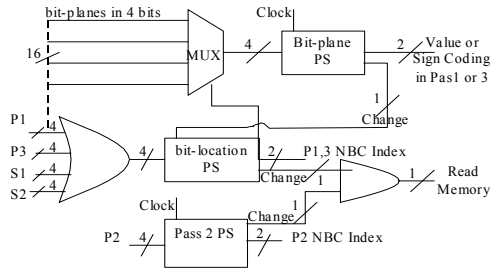

Fig.4 Parallel Pixel Skipping Unit

Pixel Skipping (PS) can provide NBC bit location in one column. NBC (Need-Be-Coded) bits in pass 2 are only obtained from the context window for pass 1 and pass 2, so it can only use one pixel skipping directly and skip non-NBC bits in one column. NBC bits in pass 1 and pass 3 are obtained from both context window for pass 1, pass 2 and context window for pass 3. Meanwhile Sign coding for one bit needs to be done immediately after value coding of the bit. So it is possible that at most four NBC bits are in the same bit location. In order to handle it, two pixel skipping units are adopted: bit-plane PS and bit-location PS. PS is a simplified pixel skipping logic gate [6].

First, NBC bits (P1, P3, S1, S3) for sign coding and value coding in pass 1 and pass 3 are ORed to 4 bits, which are supplied to bit-location PS to skip non-NBC bits in pass 1 and pass 3. The bits (P1, P3, S1, S3) in the same location for sign and value in pass 1 and pass 3 constitute 4 bit-bitplanes that indicate how many NBC bits are in the same bit location. These bit-plane are multiplexed by the bit-location index from bit-location PS. Then bit-plane PS indicates that the bit in the current bit-plane is for value coding or sign coding in pass1 or pass 3. The change signal from bit-plane PS is as clock signal of bit location PS. It notifies that it finish coding of all NBC bits in the same location. Bit-location PS provides the bit location in one column. The change signal from Bit-location PS indicates that it finish coding of all NBC bits in pass 1 and pass 3. Only the coding of all NBC bits in pass 1, pass 2 and pass 3 is finished, one read memory signal is set, then next column can be read into state variables from memories.

4.3 Pipelined Pass Switching Arithmetic Encoder

To support the parallel mode, JPEG2000 uses two key methods [7]: (1) Terminate the MQ coder at the end of each coding pass; (2) Reset the MQ coder and all context states at the beginning of each coding pass. To use less hardware and remove the memory

for high-speed outputs of two context labels, pipelined pass-switching MQ coder is adopted. There are two stages: MPS&LPS coding and normalizer&byteout. In MPS&LPS coding stage, look up table is used to implement the state machine for probability estimation. After MPS&LPS coding, registers A', C' are used to store the internal values for coding states. In normalizer&byteout stage, renormalization and byte out procedure are implemented. P1, P2 and P3 registers store the intermediate values for code register C, interval register A, counter CT, B register, Index I and MPS. The number of registers for context is 31. Pass index is used as control signal to select one for pass 1, pass 2 and pass 3.

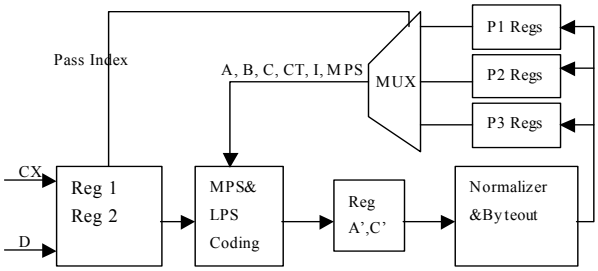

Fig.5 Block diagram of the arithmetic encoder

5. SIMULATION RESULTS

To prove the gain from this architecture, a MATLAB program is written to calculate for each column the number of processing cycles required. 20 images are picked from image processing test images. Each image has 512*512 pixels, gray scale with 8 bit pixels. For each image, a DC shift (-128), wavelet transform for 3 levels using 5/3 filter and quantization for 256 gray levels. The image is divided into code blocks. Each code block is 16*16 pixels. The probability of the number of cycles required for coding one column is shown in Fig.6. From the results, we can calculate the time required in the architecture in [3] and the proposed one here.

$$Time = \#\ of\ code\ blocks * \#\ of\ bitplanes * \#\ of\ columns * time\ for\ encoding\ a\ column$$

In both architectures, *# of code blocks, # of bitplanes* and *# of columns* are the same. So our concern is on the time required for encoding a column. The first architecture [2] needs 4 cycles for encoding each column. In the proposed architecture, the time required is

$$Column\ encoding\ time = P_2 * 2 + P_3 * 3 + P_4 * 4$$

Where P_2, P_3, and P_4 are the probability that a column needs 2, 3, and 4 cycles for encoding one column. Using the probabilities values shown in Fig. 1, in our proposed architecture, the time required is

$$Column\ encoding\ time = 0.18 * 2 + 0.21 * 3 + 0.61 * 4 = 3.43\ cycles$$

So it is clear that the proposed architecture requires less time for encoding the column, this means increasing in the speed of the whole system. We estimate that the proposed idea will increase in the system speed by 16.6% than that for the parallel mode architecture in [3]. The results also show that the proposed architecture reduces the processing time by more than 38% compared to the serial mode architecture proposed in [2].

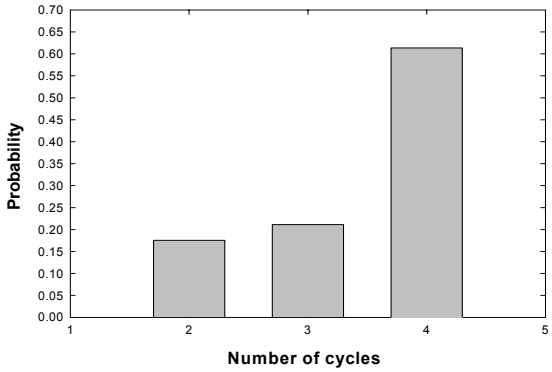

Fig.6 Probability of Number of cycles required

6. CONCLUSION

The hardware required for the proposed architecture is almost the same as the parallel mode architecture in [3]. The experimental results show that the proposed architecture reduces the processing time by more than 16.6% compared with the pass-parallel mode architecture in [3] and by more than 38% compared with serial mode architecture in [2].

ACKNOWLEDGEMENT

The authors would like to acknowledge Dr. Beth Wilson for her valuable comments and feedback. This work has been partially supported by: NSF- INT0211620, DOE EETAP- DE- FG02-97ER12220, and the Louisiana IT initiative.

REFERENCES

[1]JPEG 2000 Part I Final Committee Draft Version 1.0, ISO/IEC JTC1/SC29/WG1 N1646R, March 2000

[2]K.Chen, C.Lian, H.Chen, and L.Chen, "Analysis and Architecture Design of EBCOT for JPEG-2000", IEEE ISCAS-2001, vol.2.pp.765-768,May 2001

[3]Jen-Shiun Chiang, Yu-Sen Lin, and Chang-Yo Hsieh, " Efficient Pass-Parallel Architecture for EBCOT in JPEG2000", IEEE ISCAS-2002, May 2002

[4]D.Taubman, "High Performance Scalable Image Compression with EBCOT", IEEE Trans. Image Processing, vol.9, no.7, pp.1158-1170, July 2000

[5]D. Chai, and A. Bouzerdoum, "JPEG2000 image compression: an overview," Seventh Australian Intelligent Information Systems Conference, pp. 237-241, Nov 2001

[6]Yijun Li, Ramy E.Aly, Beth Wilson, and Magdy A.Bayoumi, "Analysis and Enhancements for EBCOT in High-Speed JPEG2000 Architectures", Proceeding of the 45th IEEE 2002 MWSCAS, Aug. 2002

[7]D.Taubman, E. Ordentkich, M.Weinberger and G.Seroussi, "Embedded Block Coding in JPEG2000", HPL_2001-35, HP Labs, Palo Alto, Feb.2001