ARCHITECTURE DESIGN OF A HIGH-PERFORMANCE DUAL-SYMBOL BINARY ARITHMETIC CODER FOR JPEG2000

Minsoo Rhu and In-Cheol Park

Korea Advanced Institute of Science and Technology (KAIST)
{minsoo.rhu@gmail.com, icpark@ee.kaist.ac.kr}

ABSTRACT

The embedded-block coding with optimized truncation (EBCOT), which consists of a bit-plane coder (BPC) and a binary arithmetic coder (BAC), is the bottleneck in realizing a high-performance JPEG2000 encoding system due to its characteristics of bit-wise processing. Although efficient architectures for BPCs have been presented, the performance of EBCOT is mainly restricted by BAC because of its sequential processing nature. In this paper, we propose a novel architecture for BAC which is based on our optimization technique named as trace pipelining. It enables parallel processing of the usual byte-out cases in BAC, and can achieve a throughput of 534 M symbols/sec, which is the highest compared to those of the previous BAC architectures.

1. INTRODUCTION

The block coding engine of JPEG2000 encoder adopts the embedded block coding with optimized truncation (EBCOT) which consists of bit-plane coder (BPC) and binary arithmetic coder (BAC). The run-time profiling in [1] showed that the EBCOT takes up more than 70% of the entire computation time on a general-purpose processor. Since the EBCOT accounts for the majority of the computation time in JPEG2000 encoding system, a high-performance EBCOT is in demand.

The BPC utilizes the neighboring information of the current bit to construct the context information consisting of context (CX) and decision (D), which will be entropy coded by the following BAC. Concerning the BPC, many architectures have been presented to output multiple D/CX pairs per cycle, but the BAC architectures presented in previous literatures could not accommodate the multiple input pairs forwarded from BPC. This is because the conventional approaches fail to consider the unique characteristics of arithmetic coding in JPEG2000, thereby significantly increasing the critical path delay and suffering from a decreased throughput.

In this paper, we propose a novel BAC architecture capable of processing two D/CX pairs per cycle with increased throughput. We perform an in-depth analysis on the unique characteristics of arithmetic coding and explain how they can be utilized to optimize BAC. The optimization techniques are examined to increase throughput and applied to actual implementation at hardware level in order to show their effectiveness.

2. BINARY ARITHMETIC CODING IN JPEG2000

The BAC adopted in the JPEG2000 standard is based on the MQ coder, which uses context-based probability estimation. During the encoding procedure, the probability interval, \( A \), and the code register, \( C \), are updated as follows: If \( D = \text{MPS} \) then \( A = A - Qe \) and \( C = C + Qe \), otherwise, \( A = Qe \) and \( C = C - Qe \) [2]. Since \( A \) must be kept in the range of \((0x8000, 0xFFFF)\), the renormalization (RENORM) procedure, shown in Fig. 1, is invoked whenever \( A \) falls below 0x8000. During RENORM, procedure BYTEOUT is invoked whenever the counter register \((CT)\) representing the available bit space in the upper 12bits of \( C \) counts down to zero. For a single D/CX pair encoding, the maximum amount of shift \((SA)\) for renormalizing \( A \) and \( C \) is 15, which is invoked when \( A = Qe = 0x0001 \). As the \( CT \) value is reset to 7 or 8 according to Fig. 1, the BYTEOUT procedure can be called twice at most during a single D/CX pair encoding, resulting in a consecutive two-byte emission.

Since the heart of arithmetic coding is composed of multiple branches and conditional statements, it is evident that the direct hardware implementation of Fig. 1 would be inefficient. Therefore, recent studies concerning BAC implementation adopt a pipelined-mechanism, [3]-[5] but the BAC is still a performance bottleneck in EBCOT. Some of the previous works adopt a dual-symbol architecture to process two D/CX pairs in a clock cycle as an effort to increase the throughput of EBCOT. However, most of them suffer from a prolonged critical path delay, because they were fundamentally cascading two single-symbol processing elements, resulting in an almost linearly increased delay. The architectures proposed in [4] and [5] have adopted some concurrency techniques in order to reduce the critical path delay, but the actual delay of them are still not as satisfactory as it should be in order to keep up with the BPC. As a result, the low performance of the BAC not only deteriorates the performance of EBCOT, but also hampers the development of the next generation BPC that may demand much higher operating frequency.
3. TRACE PIPELINING

Trace Scheduling generally refers to an optimization technique used in high-level language compilers, which separates out unlikely codes and adds handlers for exits from the likely trace. The main assumption is that the likely trace is so much more probable than the alternatives that the cost of the bookkeeping code is not a deciding factor. This approach can be also effective in hardware implementation if a certain path is far more likely to be taken than the other paths. Fig. 2 is a conceptual block diagram of how trace scheduling can be applied to improve the system performance of a pipelined architecture. Stage 1 to stage 3 are constructed under the assumption that only the usual cases, the likely traces, need to be considered within the logic path of the given pipeline stages. Other cases, the unlikely traces, are completely disregarded within these stages, so the hardware components of these stages are much more simple and compact. The primary goal in implementing these stages is to minimize the critical path delay and increase system throughput, because the majority of the execution time is occupied by processing the likely traces. The Stall-detection units, which are placed in each pipeline stage, check for any circumstances that require the pipeline stages to stall and handover the processing sequence to the auxiliary units when the unlikely cases actually occurs. The main goal in the implementation of the auxiliary units is to minimize hardware cost, because the probability of utilizing these auxiliary units is very unlikely. As long as the probability of likely traces far outweighs that of the unlikely traces, the overall performance of Fig. 2 is almost not deteriorated. We name this as the trace pipelining, which is applicable to all pipeline architectures that deal with a skewed-distribution of possible paths. In the following section, we explain how trace pipelining can be applied to optimize the BAC in JPEG2000.

4. PROPOSED DUAL-SYMBOL BAC ARCHITECTURE

Since the critical path of a pipelined BAC is established in the stage where the output bytes are extracted from C, it is crucial that an optimization on this stage be done in order to develop a high-performance BAC. [2] indicated that the inputs to the BAC have a highly-skewed distribution implying that RENORM rarely occurs. Our analysis also revealed that the probability of emitting two consecutive bytes is less than 0.001%. Based on this fact, we can infer that the probability of emitting (x bytes, y bytes) in a dual-symbol BAC, which can be approximately calculated as (Probability of emitting x bytes) times (Probability of emitting y bytes), would also show a highly-skewed distribution. Motivated by this observation, we have simulated various images on various sizes and accumulated the numbers of all possible combinations of byte emission (BE) cases. Some of these results are summarized in Table I. Since there are three possible BE cases in a single-symbol BAC, there are a total of nine possible BE cases in a dual-symbol BAC, which are ranging from (0,0) to (2,2). As expected, except for the (0,0), (0,1) and (1,0) cases, the probability of other six BE cases is less than 0.01% on the average. This means that even if two consecutive input symbols are encoded simultaneously, the usual amount of output bytes being emitted out of the BAC is still zero or one. Although the architecture in [5] has considered the skewed distribution of a single-symbol BAC, most of the rare cases are considered in building the BAC, because the conventional structure of a dual-symbol BAC is simply a concatenation of two single-symbol BACs. Fig. 3 shows a typical structure of the conventional pipeline stage deriving the updated

<table>
<thead>
<tr>
<th>Test Image (Size)</th>
<th>Number of input symbol pairs</th>
<th>Total number of BYTEOUT pair cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(0,0)</td>
<td>(0,1)</td>
</tr>
<tr>
<td>Cameraman-Gray</td>
<td>270097</td>
<td>227495</td>
</tr>
<tr>
<td>(256x256)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lena</td>
<td>917819</td>
<td>756968</td>
</tr>
<tr>
<td>(512x512)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Space-craft</td>
<td>49932740</td>
<td>42046721</td>
</tr>
<tr>
<td>(4096x4096)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 3. Conventional structure of a pipeline stage in a dual-symbol BAC that derives the updated value of C.
value of $C$ in a dual-symbol BAC. Because of the serial processing structure, it is not easy to eliminate the six rare cases out of the usual logical path. In order to eliminate the six rare cases out of the usual logical path, we have adopted another technique named as renormalization look-ahead (RLA). The RLA has originated from the distributive law in mathematics, and is similar to that of a carry look-ahead distribution of BE combination. Consider the case of deriving the updated value of $C$ using RLA, where we have derived an efficient way of processing BAC for the serial processing structure, it is not easy to eliminate the six rare cases out of the logical path. As inferred from Fig. 1, the renormalization of $C$ is very time-consuming because when a byte emission is triggered during the renormalization, $C$ needs to be masked by either 0x7FFFF or 0xFFFFF and go through another shifting if $SA$ is larger than $CT$. The RLA is powerful because it illuminates how to minimize the number of consecutive shift operations and calculate it in parallel.

Consider the case of deriving the updated value of $C$ ($C_{\text{updated}}$) which goes through both RENORM and BYTEOUT procedures, while encoding the first $D/CX$ ($D_1/CX_1$), $C$ is first shifted by $CT$-bits, masked, and shifted again by ($SA_1-CT$)-bits. Therefore, $C_{\text{updated}}$ can be expressed as

$$[(C_{\text{temp}}, << CT) & \{\text{MASK}, << (SA_1-CT)\}]$$

where $C_{\text{temp}}$ is determined as either $C$ or $C+Qe$, and $\text{MASK}$ refers to either 0x7FFFF or 0xFFFFF. Here, shift-left by $CT$-bits, represented by ($<<CT$), is actually equivalent to a multiplication by $2^{CT}$. Since ($<< (SA_1-CT)$) is also equivalent to a multiplication by $2^{(SA_1-CT)}$, we can simplify Eq. (1) as below using the distributive law.

$$(C_{\text{temp}}, << SA_1) & \{\text{MASK}, << (SA_1-CT)\}]$$

As the $SA$ and $CT$ are available at the start of a pipeline stage, $(SA_1-CT)$ can be calculated while $C_{\text{temp}}$ is being derived. Therefore, Eq. (2) can be calculated concurrently with only a single shift operation in the critical path. If there is no byte emission while encoding $D_2/CX_2$, the final value of $C$ in (1,0) case is obtained by deriving $C_{\text{temp}_2}$, which is determined as either (2) or (2)$+Qe$, and shifting it by $SA_2$-bits. Based on the same mechanism, the three common BE cases in a dual-symbol BAC, which accounts for 99.99% of the processing time, can be optimized for parallel processing as summarized in Table II, where subscripts have been added in the equations in order to differentiate between the values related to the encoding of $D_{1/CX_1}$ and $D_{2/CX_2}$. In the table, $\text{MASK}_1$ functions exactly the same as $\text{MASK}$ used in the first BE case. Using this approach, we can look-ahead in advance the renormalization amount required for the renormalization, thereby enabling parallel processing of BAC and decreasing the critical path delay significantly. Also, the contents in Table II can be utilized to adopt the proposed trace pipelining technique in building the BAC that cares only the three common BE cases. As mentioned before, the proposed BAC does not increase the overall cycle count noticeably due to the skewed distribution of BE combination.

5. IMPLEMENTATION OF THE DUAL-SYMBOL BAC

Since we have derived an efficient way of processing BAC for each of the BE cases, we can organize the pipeline stages considering only the usual cases. Fig. 4 shows the overall circuit diagram of the proposed architecture, which contains three main pipeline stages and three auxiliary units to compensate for the six rare cases.
rare BE cases. As the critical path is established in stage 3, stage 1 and stage 2 are implemented similar to the conventional dual-symbol BACs. Stage 3 is organized under the assumption that only a single output byte can be emitted out of BAC. Therefore, it only considers the BE cases of (0,0), (0,1) and (1,0), leaving the rest of six BE cases to the auxiliary units. During the three usual BE cases, which amount to more than 99.99% of the execution time, the auxiliary unit is in the IDLE state and stage 3 solely derives the output bytes for the six rare cases, and are optimized using RLA as in stage 3. Since the probability of BE decreases as the number of output bytes increases from (0,0) to (2,2), we have organized the auxiliary units as shown in Fig. 4.

As the possibility of the six rare cases is extremely low, the total number of stalled cycles is very small, meaning that the overall performance of the proposed BAC remains intact. Also, since only a single output byte can be emitted per cycle, there is no need to add an additional FIFO which is usually required in conventional architectures.

5.1 Minimizing hardware cost in the auxiliary units

According to Fig. 6, a maximum amount of three cycles are required for the (2,2) BE case. Yet, according to the trace pipelining concept, the throughput of BAC would not be deteriorated even if we spend more cycles to handle the rare cases. Since stage 1, 2 and 3 are in the IDLE state while the auxiliary unit is operating, we exploit the inactive hardware units of these stages to minimize the hardware cost in the auxiliary units. Therefore, the actual implementation of the proposed BAC has added a few additional stall cycles in order to reduce the hardware cost of the auxiliary units as much as possible.

6. PERFORMANCE RESULTS

The proposed architecture has been described in Verilog HDL and synthesized using Synopsys Design Compiler with TSMC 0.18-μm cell library. Table III compares the proposed architecture to previous literatures. To clearly visualize the effectiveness of our optimization schemes, we have implemented a reference version by simply cascading two single-symbol BACs. Results showed that compared to the reference version, the proposed techniques enable the performance to be doubled. When compared to [5], which showed the highest performance among the previous architectures, the proposed BAC shows a 26% increase in throughput as well as a 40% saving in gate area.

7. CONCLUSIONS

We have proposed an optimization scheme, named as trace pipelining, which makes the best use of the skewed-distribution of BE cases and increases the encoding throughput. Trace pipelining has been applied to make the three common cases of zero/single-byte emission run faster and rule out the six rare cases from the critical path. The RLA was utilized to further decrease the critical path delay, which was not able in previous literatures due to its serial processing nature. Compared to the conventional architectures, the proposed techniques suggest a systematic way of developing a high-performance BAC. Experimental results show that the BAC developed based on the proposed techniques can achieve a significant increase in throughput with much less hardware cost.

8. REFERENCES