FPGA Implementation of Multi-alphabet Arithmetic Coding using Rotating Intervals

Rajesh Adluri G. Surender Babu Philemon Daniel
Department of Electronics and Communication, National Institute of Technology, Hamirpur,
Hamirpur (H.P)-177005, India

ABSTRACT

We present a modified AC scheme for multi-alphabet that offers both encryption and compression. The system utilises an arithmetic coder in which the overall length within the range [0, 1] allocated to each symbol is upheld, but the conventional assumption that the orientation of the symbols remains unchanged throughout the process is removed. The encryption and compression can be carried out simultaneously, by rotating intervals that are invoked by a rotation key without affecting the efficiency. The proposed methodology also facilitates a direction key to make the rotations in either direction.

Index Terms: Arithmetic coding, cryptography, data compression, rotation.

1. INTRODUCTION

Large information storage and high data rate transmission are the requisite features of today’s multimedia systems. Data compression can effectively tackle these demands. Lossy and lossless compression are the two data compression techniques where the latter is predominantly used in medical images, test data sets, storage and retrieval of database records, as any loss in the data is unacceptable. Huffman and arithmetic coding techniques play pivotal role in the field of lossless data compression.

Huffman coding paved the way for its implementation in earlier days due to its simplicity. Nevertheless, it needs an integral number of bits to represent a symbol and performs poorly in adaptive data compression [1]. Arithmetic coding, on the other hand has the ability to translate the entire sequence into one number represented in fraction, rather than translating each symbol of the sequence into a series of digits and is able to compress close to the symbol entropy. Besides it allows adaptive compression and incremental transmission, thus permitting one to compress data on the fly.

Based on the probability of occurrence of the input symbols, arithmetic coding recursively updates the coding interval and represents the stream of input symbols with a single floating point number. Arithmetic coding (AC) typically enables very high coding efficiency since multiple symbols are coded jointly. Traditional arithmetic coding has been proved unsecured owing to the plain text attack shown by Cleary et al [2].Bergen and Hogan have considered the problem of inferring the underlying symbol probabilities and partioning of the [0, 1) interval using observations of an arithmetic encoder output [3]. Liu et al [4] presented a system using table based bit sequence substitutions to provide encryption during arithmetic coding. Grangetto et al. [5] described encryption based on random swapping of the two intervals for a binary arithmetic coder, who utilized this approach to encrypt JPEG 2000 coded images. More recently Kim et al. [6] proposed interval splitting AC approach to encrypt the data in which the intervals associated with each symbol, which are continuous in a traditional arithmetic coder can be split according to a key known to both the encoder and decoder.

We adopt an approach the intervals are rotated for each symbol corresponding to a rotation key as opposed to the traditional arithmetic coding where the interval ordering is fixed throughout the process. The orientation direction is known to both encoder and decoder prior to the commencement of the process by a direction key that specifies the direction of the rotation. The concept of interval rotating can be applied to a source having any number of different alphabets.

The rest of the paper is organized as follows: section II briefly explains the rudiments of AC and the proposed rotating arithmetic coding method. Section III deals with algorithm of proposed methodology and its practical implementation. Section IV presents efficiency, secrecy and synthesis results along with related discussions. Finally, section V concludes the paper.

2. ROTATING ARITHMETIC CODING

For simplicity, we carry out the future discussions for the source with four symbols A, B, C, D. The traditional AC is a recursive process, according to the symbol probabilities the interval [0, 1) is partitioned. No matter which symbol interval comes first and last , it does not affects the coding efficiency because the final sub interval length that decides the coding efficiency will be same, provided that the interval order is maintained unique throughout the process. Conversely, the proposed rotating arithmetic coding rotates encoding intervals based on a rotation key. The rotation key decides whether a rotation is to be followed or not. The total length of the rotation key is lesser than the total number of symbols to be encoded by one bit. For instance, if there is a sequence that has a total of ‘N’ symbols to be encoded, the length of the rotation key is taken as N-1bits.The encoding is done symbol by symbol basis. For each symbol, the rotation key decides whether or not to make a rotation with an exception for first symbol. The working of the conventional AC and the proposed rotational AC is briefed up in the following example and processes sketched in Fig .1 (a) and Fig.1 (b)

Consider a source string ABCD with probabilities p (A) =1/4, p (B) = 1/4, p(C) = 1/4, p (D) =1/4, and rotation key as 101. For this case direction key is set to 0 indicating leftward rotation of the intervals.

TI is the final sub interval of the conventional AC from which a real number to be sent to the decoder to decode the input string. RI is the final sub-interval for the rotational arithmetic coding by encoding one symbol at a time. Note that TI and RI have the same length. If we decode the output codeword of the rotational arithmetic coding with conventional arithmetic coding procedure we get output as AA for first two symbols third symbol and fourth symbol depends on interval number chosen by Rotating AC.

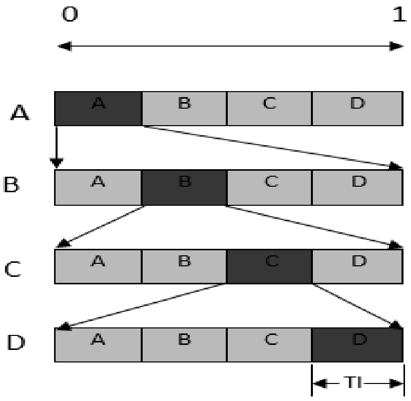

Fig. 1 (a) Conventional arithmetic coding

Only when there is a perfect synchronization of encoder‘s direction and rotation keys with the decoder's, correct input stream shall be produced from coded output word. It is to be observed that both the direction and rotation keys are known to encoder and decoder prior to the initiation of the process without transmission. Rotating AC generalizes to conventional AC if all the bits of the rotation keys are set to zeroes irrespective of direction key. When all the bits of rotation key are set to one, the number of rotation of intervals is maximum in the direction specified by the direction key.

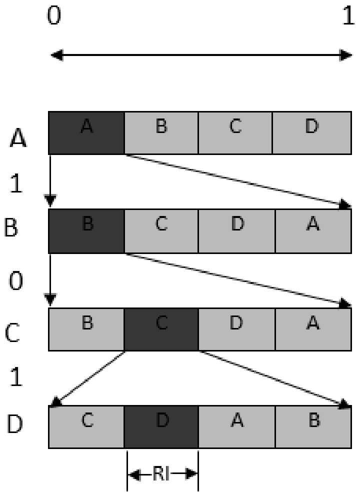

Fig. 1 (b) Conventional arithmetic coding

3. IMPLEMENTATION

In this section we describe briefly practical implementation of Rotating AC by considering the issues of Conventional AC. In conventional arithmetic coding, a sequence made of 'm' multiple symbols i.e., a = (a_1, a_2, a_3,……, a_m) is encoded to a fractional number in the interval [0, 1) .While encoding symbols, until we meet an EOF (end of file) symbol or satisfy the symbol count, recursively compute the sub interval corresponding to the symbol a_i (where 1 $\leq$ i $\leq$ T) that actually occurs. To find out the sub intervals of the corresponding symbols that actually occurs, two cumulative probabilities need namely cumulative probability of proceeding symbol

$$P_p = \sum_{k=1}^{i-1} p_k$$

and cumulative probabilities of the current occurring symbol $P_c = \sum_{k=1}^{i} p_k$ then the new sub interval is

$$[L + P_P(H - L), L + P_C(H - L)) \qquad (1)$$

where L and H indicates the lower and higher limits of the sub interval respectively with initial values being 0 and 1.We get the cumulative probabilities from the model that may be adaptive (dynamically estimating the probability of each symbol based on all symbols that precede it) static (using a preliminary pass of the input sequence together statics) or fixed (using fixed probabilities for all sequences to be encoded). The basic implementation of arithmetic coding recursive approach above described has two major difficulties: the shrinking interval requires the use of high precision arithmetic, and no output is produced until the entire symbol sequence has been read.

A. Limited precision and incremental transmission

The solution to the shrinking sub interval is to rescale them as the sequence is being coded. Rescaling allows, using a limited precision to code the entire sequence. However this process must be implemented such that the previously calculated portions of the code word are not lost. The solution to the preserving previously calculated portions of the coded word is to transmit each leading output bit as soon as it new subinterval. Rescaling and incremental transmission make the basic conventional AC to check three cases after calculating new subinterval.

Case 1: If the new sub interval lies entirely within $[0,0.5)$, transmit output as 0 and if there are any bits to follow (BF) transmit 1, then rescale the sub interval as $[2L_S, 2H_S)$ .where L_S and H_S are the lower and higher values of the sub interval.

Case 2: If the new sub interval lies entirely within $[0.5,1)$, transmit output as 1 and if there are any bits to follow (BF) transmit 0,then rescale the sub interval as $[2(L_S - 0.5), 2(H_S - 0.5))$.

Case 3: If the new sub interval lies within $[0.25,0.75)$, we don't yet know the next output bit but we know the bits to follow have the opposite value. We keep track of this fact for future output by incrementing the BF count. Then we symmetrically expand the interval around 0.5 as

$$[(2(L_S - 0.25), 2(H_S - 0.25)).$$

B. Integer arithmetic

In practice the arithmetic can be done by storing the end points of the current interval as sufficient large numbers rather than denoting them by floating point or exact rational numbers, since floating point arithmetic requires complex hardware and processing. In integer implementation, consider word length having W_L in bits. The value zero is mapped to 00……..0(with W_L 0's), the value of 1 is mapped to 11…..1(with W_L 1's).The value 0.5(HALF) is mapped to 10…..0(with $W_L - 1$ 0's), 0.25(QTR) is mapped to 010….0(with $W_L - 2$ 0's). We also use integers for frequency counts used to estimate symbol probabilities. Now the cumulative probabilities can be written as $\frac{C_c(a_i)}{T}$ where $C_c(a_i)$ = $\sum_{k=1}^{i} n_k$ and n_i is the number of times symbol a_i has appeared in the sequence. Rescaling the sub interval can be performed easily with integer implementation for case 1and case2 as shift left by one bit for the multiplication by 2. Choosing the word length W_L used for the integer implementation is not a trivial task. If W_L is large compression will be affected, if W_L is small the accuracy of the sub intervals will be compromised in the execution of the algorithm. W_L is calculated according to the smallest sub interval which is encountered in the execution of the encoding and decoding algorithm [7].

$$2^{W_L} \geq 4T \qquad (2)$$

The pseudo code for the proposed rotating AC integer implementation of encoding and decoding a sequence with four different symbols is presented in Fig. 2 & Fig. 3 respectively.

4. EFFICIENCY, SECRECY AND SYNTHESIS RESULTS

A. Efficiency and secrecy

In Arithmetic coding the number of bits in the code word is decided by the final sub interval because AC is one type of variable length coding technique. In variable length coding small interval needed large number of bits to represent it uniquely and vice versa. In section II Fig. 1(a) and Fig.1 (b) shows the length of final sub intervals of conventional AC (TI) and rotational AC (RI) are the same. So, the proposed methodology has the high coding efficiency as the conventional AC.

Rotating AC requires prohibitively high computational cost to be broken. Even if the attacker gets an access to rotating AC encoder, it is require to compare AC coded and rotating AC coded sequences for a known input sequence (for e.g. all one's sequence) to extract the rotating interval decisions. The total number of trials that are require to find the rotation key with N bits is 2^N .Further the direction key increases the complexity by an order of two i.e. 2^{N+1}. If care is taken to modify the direction and rotation keys in subsequent cases, substantially higher robustness would result against the plain text attacks.

```
$L \leftarrow 0; H \leftarrow 1; BF \leftarrow 0; SC \leftarrow 0$; rotation $\leftarrow 0$, $i \leftarrow 0; k \leftarrow 0$
While $SC < T$ do
    Get next symbol in sequence $a_k$.
    $L \leftarrow [\frac{(H-L+1)C_C(a_K-1)}{T}]$;
    $H \leftarrow L + [\frac{(H-L+1)C_C(a_k)}{T}]$;
    SC ← SC + 1;
    If ( $SC > 0$ and $RK[i] = 0$ ) then
        Case (rotation)
            $00:$ rotation $\leftarrow 01$, arrange cumulative counts of the symbols, intervals to be in order IB, IC,ID,IA
            $01:$ rotation $\leftarrow 10$, arrange cumulative counts of the symbols, intervals to be in order IC,ID,IA, IB
            $10:$ rotation $\leftarrow 11$, arrange cumulative counts of the symbols, intervals to be in order ID,IA, IB, IC
            $11:$ rotation $\leftarrow 00$, arrange cumulative counts of the symbols, intervals to be in order IA, IB, IC,ID
        End case
    End if
    $i \leftarrow i+1$
    While case1 or case2 do
        If $MSB(H) = MSB(L)$ then
            store $MSB(H)$
            $L \leftarrow L << 1$;
            $H \leftarrow (H << 1) + 1$;
            While $BF \geq 1$ do
                Store $\overline{MSB(H)}$
                $BF \leftarrow BF - 1$;
            End while
        End if
        If case 3 then
            $L \leftarrow (L - QTR) << 1$
            $H \leftarrow (H - QTR) << 1 + 1$;
            $BF \leftarrow BF + 1$;
        End if
    End while
    $k \leftarrow k + 1$;
End while
```

```
$L \leftarrow 0; H \leftarrow 1; k \leftarrow 0;$ ; read $W_L$ bits of code word into $t$ ;SC ← 0; rotation ← 0;
While $SC < T$ do
    While $[\frac{(t-L+1)T-1}{H-L+1}] \geq C_C(a_k)$ do
        $k \leftarrow k+1$
    End while
    Output symbol $a_k$
    $L \leftarrow [\frac{(H-L+1)C_C(a_K-1)}{T}]$;
    $H \leftarrow L + [\frac{(H-L+1)C_C(a_k)}{T}]$;
    SC ← SC + 1;
    If ( $SC > 0$ and $RK[i] = 0$ ) then
        Case (rotation)
            $00:$ rotation $\leftarrow 01$, arrange cumulative counts of the symbols, intervals to be in order IB, IC,ID,IA.
            $01:$ rotation $\leftarrow 10$, arrange cumulative counts of the symbols, intervals to be in order IC,ID,IA, IB.
            $10:$ rotation $\leftarrow 11$, arrange cumulative counts of the symbols, intervals to be in order ID,IA, IB, IC.
            $11:$ rotation $\leftarrow 00$, arrange cumulative counts of the symbols, intervals to be in order IA, IB, IC,ID.
        End case
    End if
    $i \leftarrow i+1$
    While case1 or case2 do
        If $MSB(H) = MSB(L)$ then
            store $MSB(H)$
            $L \leftarrow L << 1$;
            $H \leftarrow (H << 1) + 1$;
            $t \leftarrow t << 1$ + next bit of code word
            While $BF \geq 1$ do
                Store $\overline{MSB(H)}$
                $BF \leftarrow BF - 1$;
            End while
        End if
        If case 3 then
            $L \leftarrow (L - QTR) << 1$
            $H \leftarrow (H - QTR) << 1 + 1$;
            $t \leftarrow t << 1$ + ( next bit of code word)
            $BF \leftarrow BF + 1$;
```

Fig.3. Decoding Algorithm

B. Synthesis results

The proposed rotating arithmetic coder for four different symbols with an input sequence length of 16 bits was synthesized using the Xilinx Virtex-4 FPGA platform. The hardware was designed in verilog hardware description language (VERILOG HDL) and modelsim was used to carry out the simulation. the unit was synthesized using devices from the XILINX virtex-4 FPGA family. table I reports the device utilization for realizing rotating arithmetic coder with static model. the estimated power consumption for the design was reported to be 38 mw. the hardware complexity of the proposed methodology is slightly higher as compared to conventional ac owing to the hardware required to realize rotation mechanism.

Table I
Device Utilization (Xc4vlx100)

ITEM	AMOUNT	UTILIZATION (%)
Number of slices	1466	23
Number of Flip-Flops	275	2
Number of 4 input LUTs	2714	22
Number of bonded IOBs	56	23
Number of GCLKs	1	3

4. CONCLUSION

In this paper, we have presented a modification of Arithmetic coding for multi-alphabet sequence that rotates the intervals based on rotation and direction keys. We have shown that with the proposed methodology, the coding efficiency compared to the conventional arithmetic coding remains unaffected. We analyzed the cost to break the codeword. To have a realistic platform the proposed design is synthesised for Xilinx field programmable gate arrays We have focused on the four multi-alphabet sequence, the method presented here can also be applied to M-alphabet and adaptive arithmetic coding.

5. REFERENCES

[1] G. Langdon and J. Rissanen, “Compression of Black-White Images with Arithmetic Coding,” IEEE Trans. Commun., vol. 29, pp. 858–867, 1981.

[2] J. Cleary, S. Irvine, and I. Rinsma-Melchert, “On the insecurity of arithmetic coding,” Computers and Security, vol. 14, pp. 167–180, 1995.

[3] Bergen and J. Hogan, “A chosen plaintext attack on an adaptive arithmetic coding algorithm,” Computers and Security, vol. 12, pp. 157–167, 1993

[4] Liu and L. Karem, “Mutual Information-Based Analysis of JPEG2000 Contexts,” IEEE Trans. Image Process, vol. 14, no. 4, pp. 411–422, 2005.

[5] M. Grangetto, A. Grosso, and E. Magli, “Selective encryption of JPEG2000 images by means of randomized arithmetic coding,” in Proc. IEEE 6th Workshop on Multimedia Signal Processing. Siena, Italy, pp. 347–350, Sept.2004.

[6] Hyngjin Kim,Jiangtao Wen and John D villasenor,”Security arithmeticcoding”IEEE Tans.Signal processing,vol55,No.5,pp.2263-2272,2007.

[7] H.Hashempour and F.Lombardi,”Compression of vlsi test data”,In Proc.IEEE 19th International symposium on DFT,04.

[8] T. Cover and J. Thomas, Elements of Information Theory. John Wiley & Sons, 1991.

[9] Liu and L. Karem, “Mutual Information-Based Analysis of JPEG2000 Contexts,” IEEE Trans. Image Process., vol. 14, no. 4, pp. 411–422, 2005.