Advances in Engineering Innovation (2024) Volume 9 EWA Publishing
Published online: 31 July 2024 DOI: 10.54254/2977-3903/9/2024093

A multi-predictor based lossless ARGB texture compression algorithm and FPGA implementation

Handong Mo

College of Electronic Engineering (College of Artificial Intelligence), South China Agricultural University

mohandong@foxmail.com

Abstract. In the fields of GPU and AI chip design, the frequent read and write operations on the color buffer data (ARGB), which are intensive in graphical and image access, significantly impact performance. There is a need for applications that require random access and only read small images once. To address this situation, this paper proposes an algorithm with lower modeling complexity, yet achieving near-complex implementation results, along with its FPGA implementation method. Through actual testing on multiple images, the average lossless compression rate reached 40.3%. With hardware acceleration, the execution efficiency of the algorithm was further improved, ensuring both compression rate and speed, thus confirming the effectiveness of the algorithm.

Keywords: lossless compression, image compression, texture compression, FPGA

1. Introduction

In the field of lossless image compression, numerous techniques have been developed. Reference [1] uses methods related to the Deflate algorithm to compress images block by block, reference [2] proposes a lossless image compression method based on a variable bit rate block coding acyclic graph model, and reference [3] proposes a lossless image compression method based on local texture features. Ultimately, lossless image compression technologies are primarily framed by predictors, feedback mechanisms, and entropy encoders. Among these, JPEG-LS, JPEG2000, CALIC, and the latest lossless image compression technology FLIF all follow this framework [4][5][6].

In response to the above situation, the algorithm designed in this study adopts a block compression strategy, which divides a complete image into multiple 8×8 blocks for compression, using a predictor with sufficiently low computational complexity and Adaptive Arithmetic Coding (AAC) to ensure the algorithm is hardware-friendly while maintaining the compression rate, to meet the stringent requirements for image read/write speed and power in graphic rendering and deep learning training.

2. Design of the lossless compression algorithm

The compression algorithm proposed in this paper follows these steps:

(1) Divide the image into multiple 8×8 tiles in raster scan order; the subsequent algorithm compresses each tile individually.

(2) Select an appropriate predictor based on the position of the pixel, and calculate the residuals.

(3) Determine if there are negative domain residuals. If there are negative domain residuals, remap them to the positive domain; if not, proceed directly to entropy coding.

(4) Use adaptive arithmetic coding to encode the residuals, resulting in a shorter binary bitstream.

(5) Check if the binary bitstream from the entropy encoder has completed updating the multi-symbol probability table. If not, continue with another round of entropy coding; if so, obtain the final compressed bitstream.

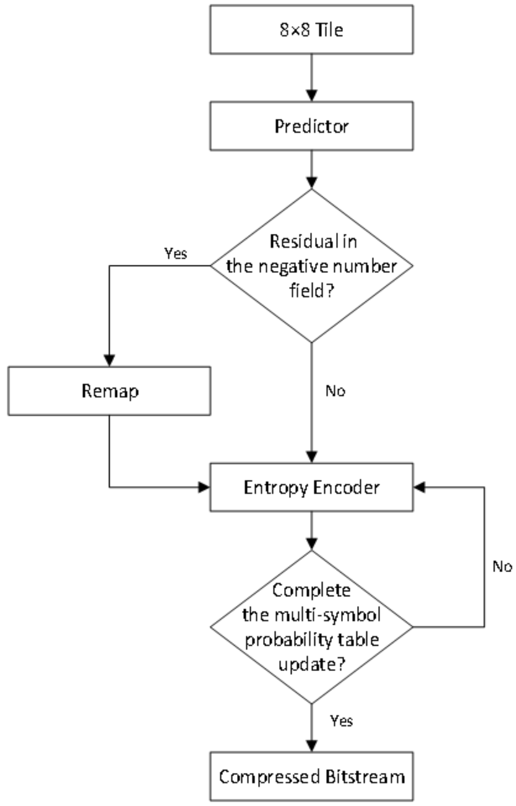

Figure 1. Flowchart of the Compression Algorithm Implementation

Note: The illustration from top to bottom sequentially represents: after the 8×8 tile enters the predictor, determine if there are negative domain residuals. If there are negative domain residuals, they need to be remapped to the positive domain before entering the entropy encoder. If not, they can be directly input to the entropy encoder. After entering the entropy encoder, roughly determine if the multi-symbol probability table update is complete. If not, return to the entropy encoder step. If it is complete, proceed to the final step of the compressed bitstream.

3. Implementation of the lossless compression algorithm

3.1. Selection of predictors

Different predictors are selected based on the position of the pixel within the block to calculate the pixel residuals, which is the difference between the predicted pixel value and the original pixel value. The selection of predictors is shown in the figure.

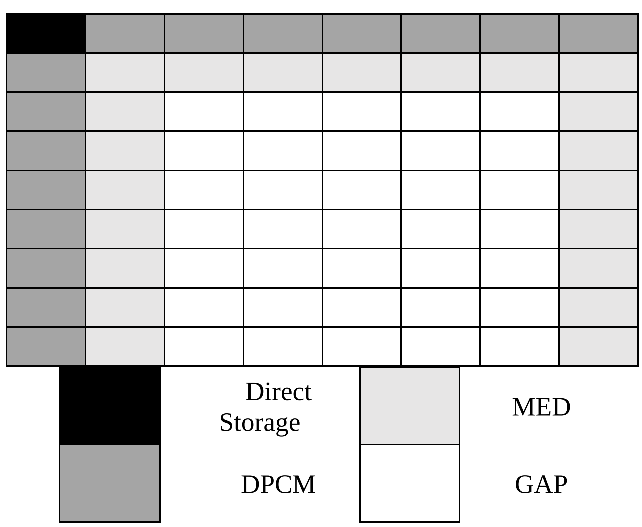

Figure 2. Different Predictors Used for Different Pixels within an 8*8 Block

3.2. DPCM

Differential Pulse Code Modulation (DPCM) is a digital signal encoding method that encodes the difference between consecutive sampling points. This difference is usually smaller than the amplitude of the original signal, thus it can be represented with fewer bits, thereby achieving data compression [7]. Compared to other complex encoding methods, DPCM is relatively simple to implement, has low computational complexity, and is suitable for real-time processing. However, its performance is not ideal for images with complex textures. The calculation formula for DPCM is as follows:

$$\hat{x} = \begin{cases} x_{i-1,j}, if\ h = 0 \\ x_{i,j-1}, if\ w = 0 \end{cases}$$

Where, x is the original pixel value, $\hat{x}$ is the predicted pixel value, and i and j are the horizontal and vertical coordinates of the image block width w and height h, respectively.

3.3. MED

The basic idea of the MED (Median Edge Detector) predictor is to predict pixel values by calculating the median of the neighboring pixel values of the current pixel, combined with gradient information. As a median predictor, the value of MED does not exceed the range of a, b, and c, and it provides a prediction system that detects past and upcoming edges. Although its ability to predict sharp edges is limited, in practice, it is sufficient to be used as a common predictor. The calculation formula for the MED median predictor is as follows:

$$\hat{x} = \begin{cases} min(a,b), & if\ c \geq max(a,b) \\ max(a,b), & if\ c \leq min(a,b) \\ a+b-c\ , & otherwise \end{cases}$$

c	b
a	x

Figure 3. Reference Pixels for MED

3.4. GAP

To further improve prediction accuracy without significantly increasing resource consumption, we use a more precise predictor, GAP (Gradient Adjusted Prediction). GAP prediction is primarily based on gradient information. To achieve this, its reference pixel distribution is more extensive, as shown in Figure 4.

		I_{nn}	I_{nne}
	I_{nw}	I_n	I_{ne}
I_{ww}	I_w	I	

Figure 4. Reference Pixels for GAP

The modeling process of GAP is as follows:

$$d_h = |I_w - I_{ww}| + |I_n - I_{nw}| + |I_{ne} - I_n|$$
$$d_v = |I_w - I_{nw}| + |I_n - I_{nn}| + |I_{ne} - I_{nne}|$$

IF$(d_v - d_h > 80)${Severe horizontal edge} $\hat{I} = I_w$
ELSE IF $(d_v - d_h < -80)${Severe vertical edge} $\hat{I} = I_n$
ELSE {
    $\hat{I} = (I_w + I_n)/2 + (I_{ne} - I_{nw})/4$
    IF$(d_v - d_h > 32)${Horizontal edge} $\hat{I} = (\hat{I} + I_w)/2$
    ELSE IF$(d_v - d_h > 8)${Weak horizontal edge} $\hat{I} = (3\hat{I} + I_w)/4$
    ELSE IF$(d_v - d_h < -32)${Vertical edge} $\hat{I} = (\hat{I} + I_n)/2$

ELSE IF($d_v - d_h < -8$){Weak vertical edge} $\hat{I} = (3\hat{I} + I_n)/4$
}

3.5. Residual remapping

Before entropy coding, the range of the predicted residual $\tilde{I}$ is $[-255, 255]$. To minimize the number of symbols, it is necessary to remap the predicted residuals in the negative domain to the positive integer domain. When $\tilde{I} \le 2^{z-1} = 128$, the potential prediction errors are rearranged as follows:

$$[-\tilde{I}, \ldots, 0, 1, \ldots, \tilde{I}, \tilde{I}+1, \ldots, 2^z - 1 - \tilde{I}]$$
$$\Rightarrow [0, +1, -1, \ldots, +\tilde{I}, -\tilde{I}, \quad \tilde{I}+1, \tilde{I}+2, \ldots, 2^z - 1 - \tilde{I}]$$

3.6. Adaptive arithmetic coding

Arithmetic coding is a lossless data compression method that can significantly reduce signal redundancy. Its high compression efficiency makes it widely used in image compression [8]. There are two coding modes for arithmetic coding: a fixed mode based on probability statistics and an adaptive mode [9]. To achieve higher compression rates, this paper adopts the adaptive mode of arithmetic coding. After remapping the prediction residuals, a residual distribution model suitable for entropy coding is obtained. The predicted residuals of each Tile will mainly be distributed in the fixed range of [0, 255]. Therefore, we first divide the symbol table of [0, 255] into 5 intervals as independent symbol probability models, and an escape symbol table of 5 characters within the range [0, 4]. The escape symbol table's role is to indicate the transition of the predicted residual to the correct coding interval when it is not within the highest probability interval and to adaptively model the highest probability interval. First, the division of the [0, 255] range is as follows:

{0,8}、{9,24}、{25,51}、{52,107}、{108,255}

The updating rules are shown in the following flowchart:

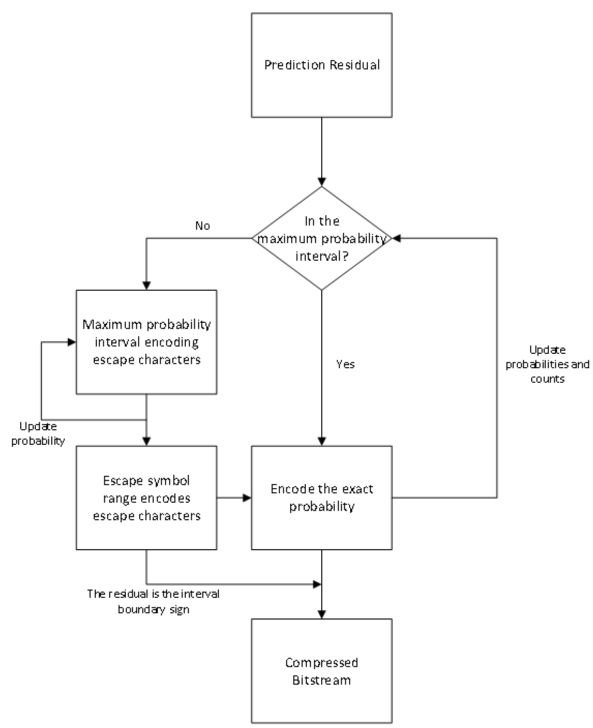

Figure 5. Updating Logic of Adaptive Probability Intervals

When encoding each predicted residual, first determine if the residual falls within the maximum probability interval. If true, encode directly and update the escape symbol count as the maximum probability judgment, while also updating the interval frequency table and escape character frequency table. If false, encode the upper or lower boundary symbol of the symbol interval based on whether the residual is greater than or less than, acting as the escape character and updating the probability. Next, accurately encode the interval within the escape character interval, updating the probability and increasing the escape symbol count, before aligning with the direct encoding scheme. It's important to note that when encoding boundary symbols of the interval, to reduce encoding of escape characters, base encoding and decoding on whether the maximum probability interval includes the boundary symbols, avoiding redundant encoding in the probability table. Also, escape characters are not encoded when encoding 0 and 255.

Arithmetic coding compresses a string of numbers between [0, 1] using probabilities and the values of upper and lower bounds, mapping the string to a decimal. To optimize for hardware implementation and avoid excessive resource use due to floating-point calculations, this paper uses integer arithmetic coding. In integer arithmetic coding, initialize the upper bound ‘high’ to 0xFFFF_FFFF and the lower bound ‘low’ to 0x0000_0000. The formula for probability calculation is as follows:

$$range = high - low + 1$$

$$high = low + (range \times high.p)/sum.p - 1$$

$$low = low + (range \times low.p)/sum.p$$

Through the calculation of the formula, we can observe that with each update, ‘high’ gradually becomes smaller while ‘low’ becomes larger, but ‘high’ always remains greater than ‘low’. When the highest bit of ‘high’ changes from 1 to 0, or the highest bit of ‘low’ changes from 0 to 1, outputting the highest bit will yield the compressed binary data.

4. FPGA system implementation

4.1. FPGA design method

We read a complete BMP image from the host computer and store it on an SD card. On the FPGA development board, we write the complete image stored on the SD card into the DDR memory. A tile of data is then transferred to the compression unit. The data from the tile transferred to the compression unit first passes through a predictor module to obtain the predicted value for each pixel. The predicted pixel values are then passed to a residual calculation module, which computes the prediction error for each pixel by comparing it with the original pixel data. The range of residual values calculated by the residual calculation module is between -255 and 255. To facilitate encoding with an adaptive arithmetic coder, we remap the predicted residuals to a predefined range([0,255]) and pass the remapped predicted residuals to the adaptive arithmetic coding module for encoding. The binary stream obtained after encoding is transferred back to DDR memory.

4.2. FPGA verification

The development platform used in this paper is Vivado 2020.1, employing the Zynq UltraScale+ series FPGA chip. By reading data via ILA and using the Vitis debugging tool Monitors to inspect data in DDR, we confirm that the hardware implementation is consistent with the software implementation. Figure 6 shows the compressed image data of an 8*8 tile, with spatial redundancy represented by zeros.

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000000010000E70	9D	BF	11	0C	00	F5	10	0B	00	F6	0F	0A	00	F7	0F	0B
0000000010000E80	00	F6	0E	0A	00	F7	1B	27	34	E3	3E	7C	D0	AA	71	ED
0000000010000E90	FF	70	73	ED	FF	6F	45	60	A7	B6	4A	64	AD	B3	45	61
0000000010000EA0	A8	B6	40	5A	9A	BB	3D	56	94	BE	40	5A	99	BC	41	5C
0000000010000EB0	9E	BA	48	66	AF	B3	51	72	C0	AA	5B	7D	CE	A2	69	83
0000000010000EC0	D2	9B	6E	84	CC	9A	6C	7F	C4	9D	6A	7B	B4	A1	6B	82
0000000010000ED0	C1	9D	70	8C	CF	96	75	94	E3	90	78	9D	F9	89	7B	A6
0000000010000EE0	FF	85	80	B6	FF	7E	82	C9	FF	77	7A	C7	FF	7A	6E	BA
0000000010000EF0	FF	82	60	A7	FF	8C	53	8F	FF	98	46	73	DD	AA	40	67
0000000010000F00	C2	B3	3F	64	BB	B5	49	78	00	00	00	00	00	00	00	00
0000000010000F10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000010000F20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000010000F30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000010000F40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000010000F50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0000000010000F60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Figure 6. Compressed Data of an 8*8 Tile

5. Test results

Table 1. Compression Ratio of Test Images

Image	Compression Ratio
sample01	6.39341%
sample02	14.4655%
sample03	37.4717%
sample04	44.8294%
sample05	37.4966%
sample06	51.6057%
sample07	55.7127%
sample08	59.2156%
sample09	61.8144%
sample10	27.6704%
sample11	24.9482%
sample12	40.7329%
sample13	59.6722%
sample14	27.246%
sample15	55.4999%
sample16	31.5251%
sample17	43.0409%
sample18	23.7606%
sample19	46.8579%
sample20	39.366%
sample21	17.053%
sample22	12.0119%
sample23	18.1744%
sample24	15.639%
sample25	48.9125%
sample26	66.2086%
sample27	53.7823%
sample28	44.7829%
sample29	58.9855%
sample30	54.9254%
sample31	57.7403%
sample32	24.2626%
sample33	68.1198%
Average	40.3007%

6. Conclusion

This paper proposes a multi-predictor-based ARGB lossless texture compression algorithm. To facilitate hardware implementation, integer adaptive arithmetic coding is employed, achieving high compression ratios under low complexity. Consistency between software implementation and hardware verification validates the correctness and effectiveness of the algorithm, providing new insights for further research in the field of lossless image compression.

References

[1] Yin, M., & Sun, G. (2024). FPGA design of a lossless ARGB data compression and decompression algorithm. *Computer Measurement & Control, 32*(02), 317-324. https://doi.org/10.16526/j.cnki.11-4762/tp.2024.02.045

[2] Chen, D., Yu, M., Dai, M., et al. (2020). Lossless image compression of acyclic graphs based on variable bit-rate coding. *Control Engineering, 27*(05), 812-818. https://doi.org/10.14107/j.cnki.kzgc.20190452

[3] Jiang, H., & Zhou, X. (2003). Lossless image compression based on local texture features. *Journal of Beijing University of Aeronautics and Astronautics, (06),* 505-508. https://doi.org/10.13700/j.bh.1001-5965.2003.06.009

[4] Weinberger, M. J., Seroussi, G., & Sapiro, G. (2000). The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS. *IEEE Transactions on Image Processing, 9*(8), 1209-1224.

[5] Sheikh, H. R., Bovik, A. C., & Cormack, L. (2005). No-reference quality assessment using natural scene statistics: JPEG2000. *IEEE Transactions on Image Processing, 14*(11), 1918-1927.

[6] Wu, X., & Memon, N. (2000). Context-based lossless interband compression--extending CALIC. *IEEE Transactions on Image Processing, 9*(6), 994-1001.

[7] Wu, X. (1997). Lossless compression of continuous-tone images via context selection, quantization, and modeling. *IEEE Transactions on Image Processing, 6*(5), 656-664.

[8] Wu, X. (2017). Research on arithmetic coding algorithm in image compression. *Computer and Digital Engineering, 45*(09), 1863-1865.

[9] Wang, C., & Wang, J. (2004). Data compression method based on energy threshold and adaptive arithmetic coding. *Power System Automation, (24)*, 56-60.