International Journal of Computer Theory and Engineering, Vol.2, No.6, December, 2010
1793-8201

On-line Image Compression based on Pipelined Architecture

Subarna Bhattacharjee, Dipak Kumar Kole and Amiya Halder

***Abstract*—Successes of multimedia applications demand on-line transmission of image data. This in turn demands on-line compression/decompression of image data files. This paper reports a pipelined architecture that can support on-line applications of image data. Spatial redundancy of an image data file are detected and removed with a simple and elegant scheme that can be easily implemented on a pipelined hardware. The scheme provides the user with the facility of trading off the image quality with the compression ratio. The basic theory of byte error correcting code (ECC) is employed in this work to compress a pixel row with reference to its adjacent row. A simple scheme is developed to encode pixel rows of a monochrome image.**

***Index Terms*—Image Compression, Check Byte, Augmented Image Creation, Partition Size, Error Encoder.**

I. Introduction

An image is a two dimensional object that provides visual information such as colour, depth, and contrast and so on to the human eyes. A digital image is obtained by sampling the information at regular intervals in either direction to discretize the information. Each sample point is referred to as a pixel. The information of bits in each pixel is stored numerically in the binary base. Higher the precision value, better would be the picture quality [1,7,8,11].

Consider an image of size 512 by 512. Assuming that each pixel is stored as one byte, then the size of the picture is almost 256Kb.The time required to send such an image at the rate of 9600 bit/sec would be approximately 3 min. If we assume a data transfer rate of 1.2 megabit/sec, the transfer time would be 1.66 sec. Even this time lag is not acceptable for on line computation/communication. Reduction of this time lag would be possible only if the image can be compressed. Further, in the context of growing demand of multimedia applications, user community look for the support of speed transmission of image data files over a communication link. With the introduction of high speed communication network, the bottleneck lies in the throughput of the compression and decompression blocks. For the sake of concreteness let us assume that there exists a scheme that has the throughput of T Kb/sec with 80% compression ratio –that is, an image of size S Kb becomes .2S Kb, Transmission of this file over the network takes .2S/T sec. Data transmission rate of this network is assumed to be greater than or equal to xT (x>1). Let there be another scheme with throughput (say) xT Kb/sec with 70% compression ratio. To transmit the corresponding compressed image file, it needs .3S/xT sec. By increasing the value of x the transmission time can be reduced substantially. So to meet the current demand it is worthwhile to look for image compression scheme having high throughput along with reasonable good compression ratio and acceptable image quality. The work presented in this paper precisely aim to achieve this goal.

Digital images very often have extremely high redundancy due to lesser variation of adjacent pixel values. This sort of redundancy is known as spatial redundancy. Similarity can also be noticed in certain components or bit planes of the image. For instance, let us consider an RGB (Red, Green, Blue) image of a scene lighted by red light. In this case though the red component may show good variation of brightness, the green and blue components may have extremely high similarity. This similarity is due to the spectral redundancy in the image [6,12,14]. The efficiency of the image compression algorithm depends on how fast and how best we can exploit this spectral and spatial redundancy to encode an image file.

We have developed an elegant and simple scheme to identify the spatial redundancy of image data file without sacrificing the quality. The basic concept for designing this scheme came from the discipline of Error Correcting Code [2,9] employed for reliable communication of digital data. Even though the concepts of differential (error) encoding [2,9] is not new, the implementation proposed in this paper differs significantly from the earlier works. Further, the major trust of the present work is to develop parallel image compression algorithm that can be conveniently implemented in hardware while achieving significantly higher throughput.

The basic scheme is reported in Section 2. Monochrome image compression scheme has been detailed in Sections 3 along with the pipelined architecture that can achieve the desired throughput for on line compression/ decompression of image files [13].

II. Basic Scheme of Image Compression

In an image there is usually a likelihood of high correlation between neighbouring pixels. Such correlations between pixels or a block of pixels are exploited to achieve image compression. Let us consider an image of m × n resolution - that is, there are m numbers of rows, each row containing n number of pixels in the digitally coded image. The basic approach we have formulated for compressing image has been derived from the concept of error correcting code (Reed-Solomon Code [3,9]) where the information bytes are encoded along with check bytes. At the receiving end the check bytes are recomputed to derive the syndrome; the non-zero syndrome is next used to locate and correct the error

in the encoded information. Analogously, we have formulated our compression scheme as follows under the assumption that there is likelihood of high correlation of pixel values in a column on two adjacent rows. A pixel is viewed as an information byte. A row referred to as R_0 (Reference Row) is viewed as the information sent with all 0's check byte and the R_1 (adjacent to R_0) is viewed as the information received with some errors. While R_0 is stored as it is in the compressed file, R_1 is encoded and stored as a set of check bytes only. The syndrome in this case is nothing but the check bytes of R_1 that can be used in conjunction with R_0 to restore R_1 at the decompression stage.

The raw image format stores the pixels in the form of a two dimensional array in row major order. Thus pixels on the same row are stored in contiguous locations. Also the precision of the input image file (the number of bits per pixel) is taken to be eight. This restriction is not really very severe as most image formats use precision of eight. Besides, the compression algorithm is itself independent of the precision that we use.

Two basic steps for encoding pixel rows are noted below

- For two consecutive rows we have implemented an averaging scheme which results in an augmented image.

Small changes in adjacent pixel values would go unnoticed by the human eye. Also for most applications, small difference in pixel values would not have much importance. Hence when the pixel values on two successive rows change by a small amount, the difference may be neglected without noticeable degradation of image quality. Another redundancy that we have exploited is the spatial redundancy of the image file. As the pixel values of consecutive points have a high correlation we can replace each group of four pixels by their average value. Of course this would lead to loss in sharpness of the picture. To take care of this situation, the actual pixel value is noted if its value differs appreciably from the average value.

- In the next phase, it is necessary to identify the difference in pixel values between a reference row R_0 and its adjacent row R_1 in the augmented image. Both the rows are divided into partitions of identical sizes.

$R_0 = R_{01}, R_{02}, \ldots$ and $R_1 = R_{11}, R_{12}, \ldots$

Each R_{ij} (i = 0,1,….., j = 1, 2,…….) consists of a set of pixels, each of say 8 bits. Each partition R_{1j} is encoded as the check bytes only that depend on the difference in pixel values of R_{0j} and R_{1j}. While scanning the pixel rows, we evaluate the difference in pixel values and so the errors (that is differences) and their locations get identified. Hence the check bytes for R_{1j} are noted as shown in Figure1 – half of the bytes represent error positions while other half specifies the correct pixel values for the error positions. Thus unlike error correcting code where syndrome is processed to locate and correct the errors, these information (the error position and correct pixel value) itself are used as check bytes in the present situation. The size of R_{1j} depends on the number of differences between the (average) pixel values of two successive rows - the difference is viewed as error. Analogous to t byte error correcting code [2, 3, 9] that employs 2t check bytes to locate and correct t number of errors, the partition size is so adjusted that no more than t errors can exist in R_{1j}. For the proposed image compression, the value of t is fixed as 4 after exhaustive experimentation.

Position of Error	Actual pixel values of the error position

Figure1: Error encoding of a partition R_{1j} check bytes

On exploiting the above two issues related to image data files, excellent compression ratio with a very high throughput has been achieved without sacrificing the quality. The conventional terminologies used to evaluate image quality are noted below.

In evaluating the reconstructed image quality, we make use of root-mean-square error (RMSE) and peak signal to-(reconstruction) noise (PSNR) as error metrics.

Definition 1: Denoting the original $N \times N$ images by f and the compressed-decompressed by $\hat{f}$, RMSE is given by

$$RMSE = \frac{1}{N}\sqrt{\sum_{i=1}^{N}\sum_{j=1}^{N}[f(i,j)-\hat{f}(i,j)]^2}$$

and represents the standard deviation of the error image.

Definition 2: The related measure of PSNR (in dB) is computed using

$$PSNR = 20\log\frac{255}{RMSE}$$

for an image with 8 bit (0 to255) pixels.

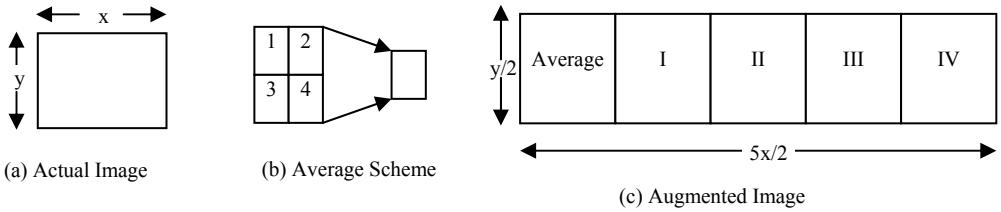
A. The Averaging Scheme

The aim of the averaging scheme (Figure 2) is to exploit the correlation between neighbouring pixels. The entire image is divided into blocks having four pixels each (Figure 2(b)). Then the augmented image (Figure 2(c)) is created as follows where corresponding to each block of four pixels there exists 5 pixels.

The actual image is divided into blocks of four pixels marked 1, 2, 3, 4. Each of the pixels of a four pixel block is kept in the corresponding segment shown in Figure 2(c). For example segment I contains pixel 1 of all the blocks. The segment II contains pixel 2 of all the blocks and so on. Thus the Figure 2(a) gets transformed augmented image as noted in Figure 2(c). The four pixels in each block are averaged and the very first pixel of the (Figure 2(c)) of the augmented image stores the average of the four pixel value. The other four pixels are now either filled with zero or the actual value of the corresponding pixels if the pixel value is significantly different from the average. If the difference is less than a specific threshold, it is put as zero, otherwise the actual pixel value is stored. To ensure that the output image quality can be adjusted as per the user's choice, the quality of threshold (QT) is set adaptively. The scheme of setting QT works as follows.

Scheme to Set QT value

We set a desired MSE (Mean Square Error) according to the quality value needed. Initially some large value is assigned to QT. After a pair of image rows is processed with the Averaging Scheme, we find the MSE of the pair of rows processed. Then we compare this MSE with the desired MSE. If the actual MSE value is greater than the desired MSE, the QT is lowered otherwise we raise the QT. This is repeated for all the rows in the augmented image to arrive at an appropriate QT value for the image file of a particular class.



(a) Actual Image (b) Average Scheme (c) Augmented Image

Figure2: Averaging Scheme

The resulting image (Figure 2(c)) that we have after the above transformation is what we will refer to future as the augmented image since an image of size $m \times n$ now becomes $1.25 \times m \times n$. If the number of edges in the image is quite low then the number of zeros in the last four segments (Figure 2(c)) would be quite high in the augmented image. This would improve the correlation between successive rows. Though we have increased the size of the image by twenty five percent, we can look forward to significant compression as there would be very high correlation between two consecutive rows of the augmented image.

B. Adaptive Partition Size

First let us define what we mean by the partition size. We have set the allowable number of errors (that is difference between a reference row and the adjacent row being encoded with check bytes) as 4. When we are encoding a row with check bytes, we can only encode for upto four differences in two successive rows. Thus when we are sending a byte string it would always be best to send a string of bytes such that exactly four of those bytes differ from the corresponding bytes on the reference row. This would lead to optimum use of the check bytes used to encode a partition. The length of the byte string being encoded that maintains this constraint of 4 errors is referred to as partition size.

From the above discussions it is evident that to get the best compression we should make the partition size adaptive, that is the number of bytes being encoded should be variable to allow optimal use of the check bytes. The first consideration is to decide on how fine a granularity to choose for the partition sizes while maintaining the above constraint. If we make the possible partition sizes too large in number then we risk a large overhead in terms of the number of bits (to indicate partition size) that we would have to include. For too little granularity in partition sizes, the problem to be faced is the inefficient use of the error correcting code. Hence it is essential to strike a balance where the overhead of the partition size does not neutralize the advantages of efficient use of the error correcting code. After exhaustive experimentation we found partition sizes as multiples of eight to be most effective.

C. Encoding of Augmented Image

After the augmented image has been constructed, the rows of augmented image are encoded. Let us denote the reference row by R_0 and the current row to be encoded as R_1. Now we will have to encode R_1 in terms of the differences of R_1 from R_0.

First we divide the pixel rows R_0 and R_1 into partitions and then encode each of the partitions of R_1. For sending each partition, one of the three following cases may arise:

1. **No Change bytes:** If the bytes in the corresponding partition of both rows R_0 and R_1 are the same, then we encode the partition of R_1 using a flag to indicate that the same bytes as in R_0 partition should be substituted.
2. **Partial Change Bytes**: If the bytes differ in less than or equal to four positions then we can encode the partition by sending the check bytes.
3. **Full Change Bytes:** If the bytes differ in more than 4, then the actual bytes of R_1 partition have to be provided to the decoder to allow it to reconstruct the partition.

It is obvious that the first situation is the most efficient that leads to maximum compression.

The tuple for **Case 1** is < Flag and Partition Size> -this is the compressed data for the partition.

Since most of the pixels for the **Case 2** do not change, the best way to encode the differences is to give location and the actual value of the pixel. As the number of differences may vary from partition to partition, the number of differences is also encoded. Hence the tuple is constructed as follows:

< Flag, Partition size, Number of errors, Location and Actual pixel value>. The size of the last field is variable and depends on the number of differences. Maximum size of the tuple is $2 + 5 + 2 + (16 \times 4) = 73$ bits. The total bit count

includes 2 bits for flag, 5 bits for partition size(since multiples of 8 upto a maximum of 256 pixels are allowed), 2 bits for number of errors and atmost 16 bits per difference. Of these 16 bits, eight are required to store correct pixel value and atmost eight bits for its location in a block of maximum 256 pixels. Exhaustive experimentation shows that the

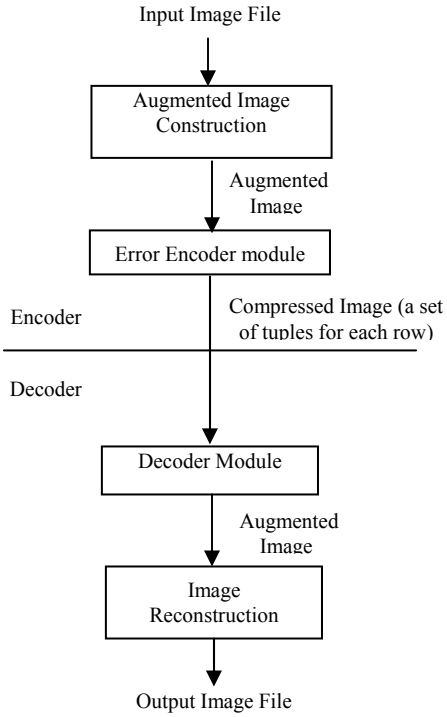


Figure 3: Block diagram of Monochrome Image Compression Scheme

number of bits required to encode a partition is usually much less than the worst case value of 73 bits.

The **Case 3** is full change encoding where the partition size is fixed as 8. To encode a partition of size eight pixels, the number of bits required is 2 + (8 x 8) = 66 bits.

The tuple is < **Flag, 8 × (Partition size)**>. Thus the third case gives the worst compression.

To achieve higher compression it is necessary to minimize the number of pixels encoded in third case. As differences occur in bursts we usually come across small regions with large number of errors and large regions with very few errors. It would be ideal to encode the relatively more error-free regions using the first two cases. The regions where large differences occur are encoded by using the third case. Hence adaptive partition size is chosen. For our implementation we have selected multiples of eight to be the partition size. To increase compression ratio, small changes in pixel values below QT (Quality Threshold) are neglected [7]. This has the advantage that correlation between rows improves. So we get higher compression ratio. However, the threshold below which we neglect changes should not be set to a high value that would lead to deteriotion of quality. Based on exhaustive experimentation we have arrived at the adaptive for a specific image. This has enabled us to provide the user with the flexibility of trading off the compression ratio with the image quality for a specific application. The next two subsequent sections describe the algorithm for monochrome image compression.

III. MONOCHROME IMAGE COMPRESSION

In monochrome images all the three colour components i.e., the red, blue and the green components are equal. This aids in compression as we can eliminate two of the components and still not lose any information. The block diagram of the monochrome image scheme is noted in Figure3.The first block is to construct augmented image. After the augmented image has been constructed the encoder block creates the tuples represent the compressed image. According to our scheme, we first try to encode using no change bytes, if that is not possible we try partial change bytes and if this also fails we encode with full change bytes.

In algorithm encode-row, len1 is the number of pixels of the augmented row which can be encoded using no change bytes. len2 is the number of pixels which can be encoded using partial change bytes. According to our scheme, we first try to encode using no change bytes, if that is not possible we try partial change bytes and if this also fails we encode with full change bytes. The sequential steps of encoding are as follows.

Algorithm 1: Algorithm for Encoding

Input: Image file

Output: Compressed image, each row encoded with a set of tuples (noted in section II.C)

Step 1. Construct Augmented image as noted in section II.C
Step 2. Encode the first row of Augmented image as it is.
Step 3. For all i > 1, encode-row R_i with. respect to R_{i-1} on executing algorithm 2.
Step 4. $R_{i-1} \leftarrow R_i$
Step 5. Go back to Step 3 till all rows are encoded
Step 6. Stop

Algorithm2 : Algorithm for encode-row

Input: A pair of rows R_{i-1} and R_i of augmented image(Figure 2(c))

Output : Encoded R_i in terms of a set of tuples, each representing a partition of the row

Step 1. Scan the pair of rows sequentially and identify each of the partitions that can be encoded with once of three tuples (Case 1, 2 or 3 as noted in section II.C);
Partitionsize ← K × 8, (K=1,2,….)
len_1←Maximum Length of string where no error present
len_2←Maximum Length of string where four errors present (Case1)
Step 2. if $len_1 > 8$ then encode Send No Change Bytes
else if $len_2 > 8$ then encode Send Partial Change Bytes (Case2)
else encode Send Full Change Bytes(Case3)
Step 3. Stop

A. Decoding Scheme

Now we present the image decompression scheme(Figure 3) which does the inverse transformation to the above compression algorithm.

The first step of image decoding consists of reconstructing the augmented image from the set of tuples of each row. This is carried out by reading the tuples and then carryout necessary inverse action to reconstruct the augmented image as per the partition size and the 'Flag' value of a tuple that specifies one of the three cases (no change, partial change and full change as noted in section II.C). The detailed block diagram of decoder module is noted in Figure 4. Assuming 4 pixels (32 bits) are read and written with memory cycle time of 40 nsec and 20 nsec clocks, the throughput of the block can be computed as follows.

For Case1, it involves transfer of data from Ri-1 to augmented row Ri through the multiplexer controlled by the output of flag decoder. Case 3 demands direct transfer of data from Ri (compressed image) to Ri Augmented image. In addition to these memory read /writes cycles, the Partial change block modifies 4 pixels to take care of four errors for Case 2—this can be completed in 4 clock cycles. Thus the maximum time taken by Case 2 can be computed as follows.

Two memory R/W = 2 × 40 = 80 nsec
Four clock cycles = 4 × 20 = 80 nsec
150% overhead for other circuit delay and associated control logic = 240 nsec
Total = 400 nsec

Hence the Decoder module of Figure 3, 32 bit of data is processed in 400 nsec resulting in a throughput of 80 Mbps for the module.

Next step consists of constructing the final image from the augmented image. In the second step the average values are first substituted then the other four segments (Figure 2(c)) of the augmented image are read. The zero pixel value of a segment is replaced by the average value and non-zero pixel values are appropriately placed to construct the final image. That is, in this step Figure 2(a) is generated out of Figure 2(c). Assuming 4 byte organized memory, the image construction phase needs 2 memory read cycle (2×40=80 nsec) to read 5 bytes of augmented image, four compare cycles (4×20=80 nsec) and one write cycle (40 nsec). Assuming 100% overhead, the total time to process 32 bits is 2×(80+80+40) = 400nsec. This again leads to a throughput of 80 Mbps. Thus the latency in the two pipelined stages of decoder is identical. The next section deals with the throughput of the encoder stage (Figure 3).

B. The detail design of encoder

The main advantage of our image compression scheme is the ease with which it can be implemented efficiently in hardware. The two basic stages of the compression hardware (Figure 3) are:

- Augmented Image Creation
- Error Encoder

To speedup the operation of our algorithm the above stages are implemented in a two stage pipeline. The basic operation of

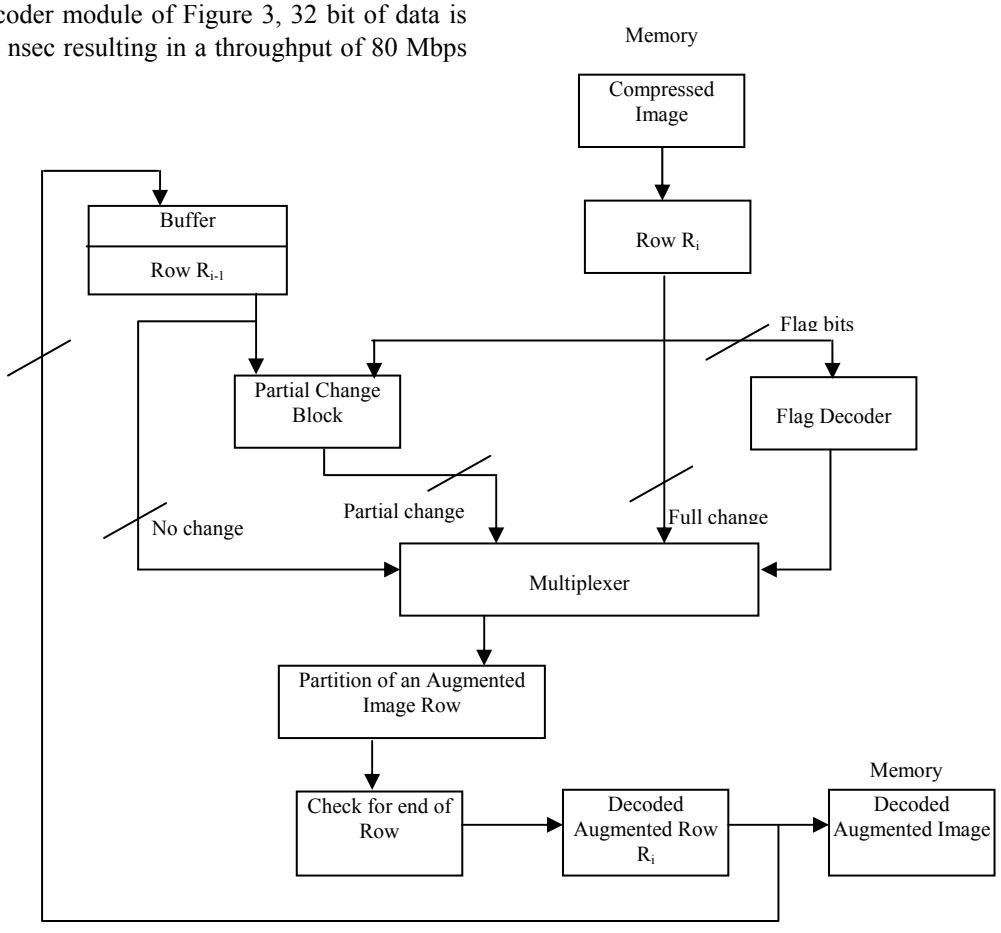


Figure 4: Block diagram of the decoder module

the pipeline is explained in Figure 5. To implement the pipeline we need two buffers for storing the results of the first stage of the pipeline. Assume that the ith augmented row is stored in the Augmented Row1 buffer. While the encoder block is encoding the ith row the Augmented row creation block generates the (i + 1)th row of the augmented image. Each of the two stages of the pipeline has been detailed in subsequent discussion.

Augmented row creation stage

The circuit diagram of the augmented image creation block is noted in Figure 6. The Row and Coloumn registers store the address of the pixel values being processed currently. The four pixels whose average is to be taken addressed using these two registers with suitable left shifts. To take the average we need to divide the sum of the values by four. This can be easily implemented by right shifting the Average register twice after the sum has been computed.

Once the average has been computed it is compared with each of the actual pixels. If the difference is greater than the quality threshold (QT), the appropriate entry of the original pixel value is made replacing zero in the augmented row. This is implemented using comparator and the two multiplexer. The multiplexer is used to determine the position where the pixel value is to be placed. The variable COLS is the half the width of the original image. Hence adding 0, COLS, 2*COLS, 3*COLS, 4*COLS the multiplexer input as offset, we get the address of the position of the pixel in the appropriate segment of the image as shown in Figure 2(c).

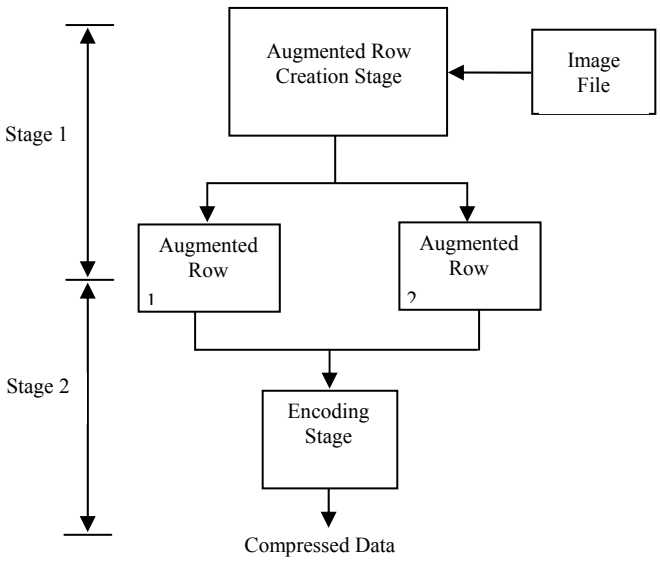


Figure 5: Pipelined Architecture for Encoder

It will be noted that the formation of the augmented row requires that each pixel value be read once from the memory. Hence the number of memory accesses required would be equal to the size of the monochrome image.

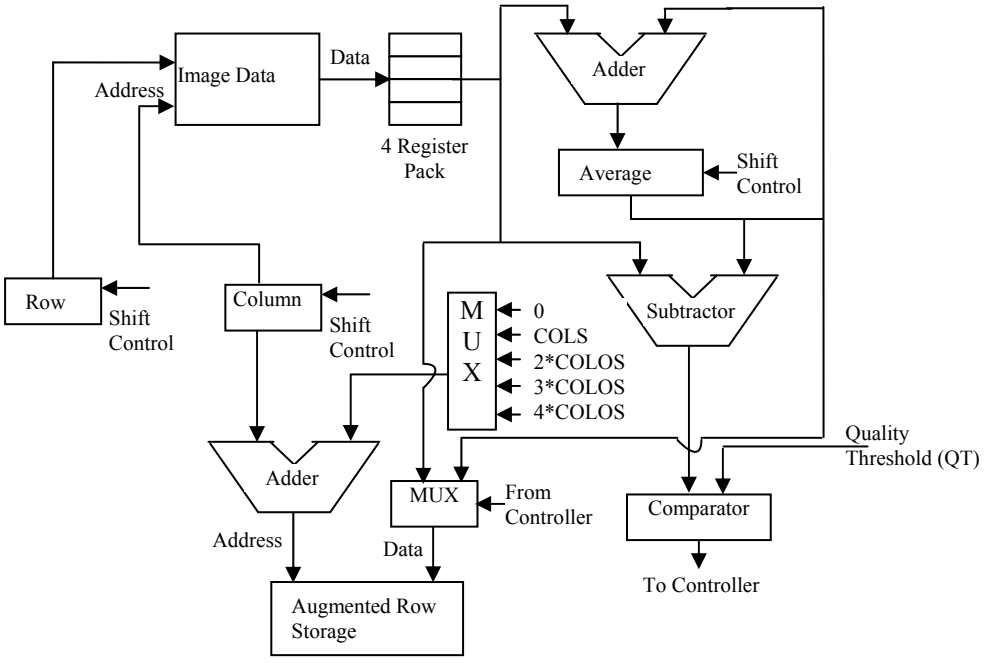


Figure 6: Augmented Image Creation Block

The time to process a block of four pixels would take four memory cycles and eleven clock cycles. The eleven cycles are needed for the following tasks (Figure 6):

- 1 clock cycle for computing average.
- 2 clock cycles (at most) for each pixel of the augmented row. Since the five values are written for every four pixels this requires 10 clock cycles.

A 50 MHz clock with 20 nsec clock cycle and 40 nsec memory cycle is assumed for the circuit. Hence time taken to process 4 bytes of input data is given by,

t1 = 4 * 40ns + 11 * 20ns = 380 nsec

Assuming 100% overhead, the time to process 32 bits is 750 nsec.

This gives a **throughput** of the Augmented Row Creation Stage as 43 Mbps. It should be noted that the estimate is a pessimistic one as we have assumed that all four pixels values differ sharply from the threshold value.

Error encoding module

The block diagram of this stage (Figure 3) is presented in Figure 7. The operation of the stage involves a set of cycles in which a tuple for a partition is generated. The quality thresholding (QT) is implemented using the subtractor and Comparator A. Counter 1 keeps track of the partition size and Counter 2 keeps track of the number of differences between the reference row and the row being encoded. Comparator B compares the of counter 2 with 0 and 4 whereby the (**Case 1, 2 or 3**) to be employed gets identified. The register bank shown stores the value of the pixels if full change (**Case 3**) encoding is to be used. In case of partial change **(Case 2**) encoding is to be used, the register bank stores the four error locations and the actual pixel values.

It should be noted that for each pixel of the augmented row being encoded we need two memory cycles. The first is needed to read in the pixels from the reference and present augmented row of augmented image storage. The second is necessary only if the difference between them is above the threshold and so we need to overwrite the pixel in the augmented row being encoded currently.

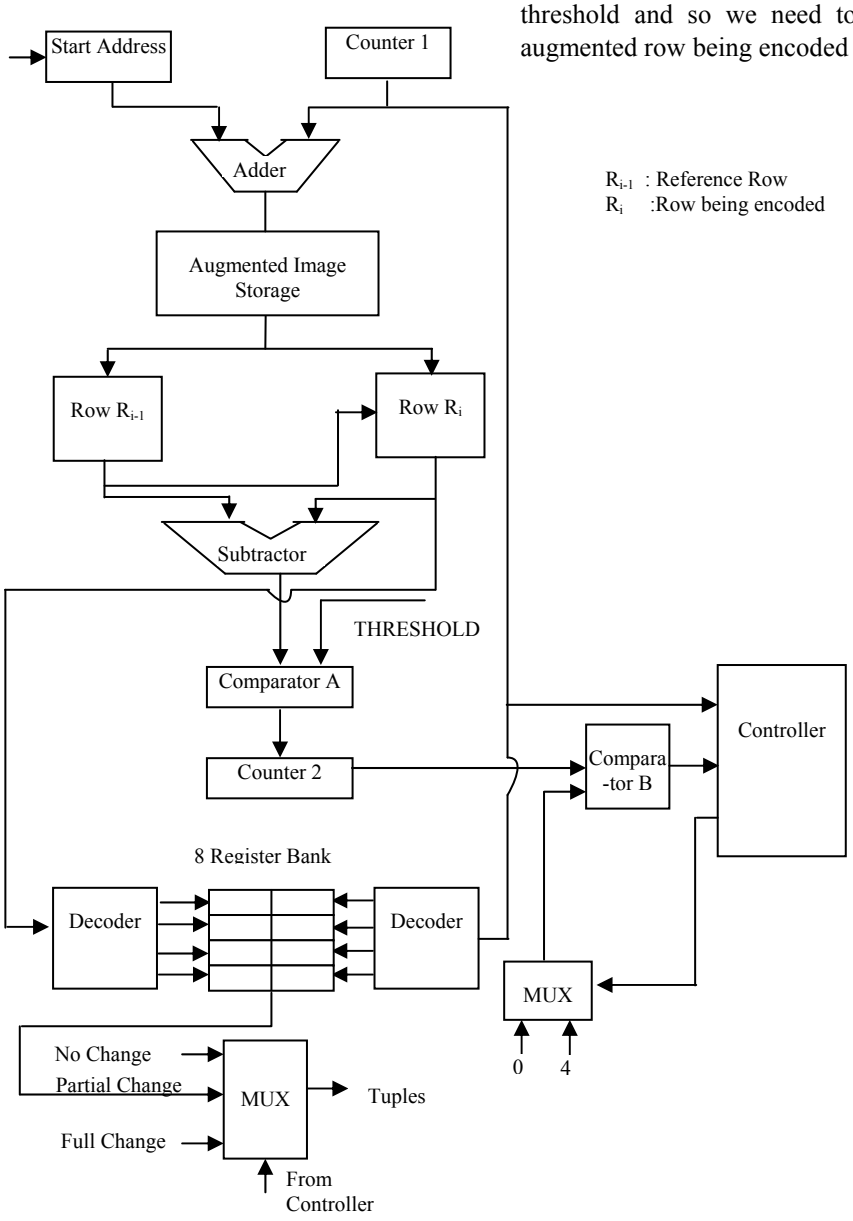


Figure 7: Architecture of the error encoder module

We have assumed that the memory clock cycle is 40 ns and the clock cycle is of 20 ns duration. We have also assumed that four clock cycles are necessary for processing each pixel-one each for subtraction, comparison in Comparator A, incrementing the counter and again comparison by Comparator B. The time taken for each pixel of augmented image creation block is given by,

$t_2 = 2 * 40ns + 4 * 20 ns = 160ns$

Hence time taken to process five pixels of the augmented row would be 800 ns. As five pixels of the augmented row correspond to four in the initial image we have a **throughput** of 40 Mbps in terms of the input data stream.

Assuming an overhead of 100% for other circuitry blocks in Figure 6 and associated control logic structure, the time to process five pixels of the augmented row to be 1600 nsec. This results in a throughput of 20 Mbps. Formation of the tuple depends on processing each pixel exactly once and therefore the **throughput** of the encoding scheme can be assumed as 20 Mbps.

We find that the encoder stage consumes a greater portion of the time and thus limits the throughput of the compression scheme. Even a fast 10 Mbps communication link would be efficiently used by the compression hardware that we have proposed. Further, with refinement of the design operated with faster clock, the throughput for the scheme can be easily pushed up to much higher level to match the higher bandwidth of the high speed communication networks of next generation

IV. Monochrome Image Compression Results

The algorithm developed has been simulated using C Program on a Silicon Graphics workstation. The input image is considered to be gray scale images [4,5,6,10]. The precision is assumed to be 8 i.e. the number of bits per pixel is 8. Hence the size of the image may be thought of as the number of pixels in the image. All the image files that we have tested are of size 256×256. The compression ratios obtained are tabulated in Table 1. The compression ratio is better than JPEG for majority of the images. However, the major motivation of this work is to design high speed pipelined hardware architecture for image compression. This has been achieved with the proposed hardware that achieves a throughput above 20 Mbps. The Circuit structure is simple and modular, and can be implemented with VLSI technology.

Quality values (PSNR) for Monochrome images:

In each of the images excepting circle and square the background shade changes continuously whereby the peak signal-to-(reconstruction) noise ratio(PSNR), as per our scheme, drops compared to JPEG results. However, for each of these images, we could not find any deterioration of the images quality so far object in the image are concerned. On the other hand for

images (say circle and square) where the background is more or less uniform and does not change continuously throughout the image, our PSNR ratio is better. After exhaustive experimentation we have set the bottom line of PSNR value as 25 at which the image quality we found good and acceptable. Also from experimental observation we confirm that for fixed background images (amber, rose, mouse) we get better compression ratio with high PSNR values.

TABLE1: Compression Ratios for Monochrome ImageS

Image	Original Size (in bytes)	Proposed scheme (size in bytes)	Proposed scheme (compression)	JPEG (size in bytes)	JPEG Compression
camera	65536	9611	0.8534	11364	0.8266
circles	65536	4364	0.9334	5854	0.9107
goldhill	65536	14234	0.7828	14263	0.7824
horiz	65536	2530	0.9614	3969	0.9394
slope	65536	5153	0.9214	5714	0.9128
squares	65536	609	0.9907	1677	0.9744
amber	64000	7426	0.8839	8489	0.8673
mouse	64000	5872	0.9082	6018	0.9059
rose	64000	4487	0.9298	6274	0.9019

V. Conclusions

An efficient scheme of Image Compression for monochrome has been introduced. The proposed scheme employs the concept of error correcting code. Compression ratio obtained by this new technique has been compared with JPEG which shows comparable compression ratio with acceptable quality. The **throughput** of the design is 20 Mbps for monochrome image compression scheme, which can match a high speed communication link and thereby supporting on-line application.

References

[1] M. Rabbani and P. W. Jones, *Digital Image Compression Techniques.* SPIE Optical Engineering Press, 1991.

[2] R. Chowdhury, I. S. Gupta, and P. P. Chaudhuri, "*CA Based Byte Error Correcting Code*," IEEE Trans. on Computers, vol. 43, pp. 371-382, march 1994.

[3] K. Y. Liu, "*Arhcitecture design for VLSI design of RS decoders*," IEEE Trans. on Computers, vol. C-33, February 1984.

[4] W. B. Pennebaker and J. L. Mitchell, "*JPEG: Still Image Data Compression Standard*", Van Nostrand Reinhold, New York, 1993.

[5] G. K. Wallace, "*The JPEG still picture compression standard*," Commun. ACM, vol. 34, pp. 31-44, April 1991.

[6] S. Bhattacharjee, S,. Das, D. Roy Choudhury and P. Pal Chouduri, "*A Pipelined Architecture Algorithm for Image Compression*", Proc. Data Compression Conference, Saltlake City, USA, March 1997.

[7] Vaseghi, Saeed V. " *Advanced Digital Signal Processing and Noise Reduction".*

[8] Pratt, William K. " *Digital Image Processing".*

[9] Moon, Toddk " *Error Correcting Coding – Mathematical Methods and Algorithms"* .

[10] Acharya, Tinku/Tsai, Ping-Sing "*JPEG 2000 Standard for Image Compression-Concepts, Algorithms and VLSI Architecture*."

[11] Barrett, Harrison H./Myers, Kyle "*Foundations of Image Science".*

[12] H. Sanchez-Cruz and R. M. Rodriquez-Dagnino, "Compression bilevel images by means of a 3-bit chain code," SPIE Optical Engineering 44-9(2005)1-8.

[13] Jorg Ritter and Paul Molitor, " *A pipelined architecture for partitioned DWT based lossy image compression using FPGA's*," International Symposium on FPGA, pages 201-206,2001.

[14] Amiya Halder, Dipak Kumar Kole and Subarna Bhattacharjee*," On-line Colour Image Compression based on Pipelined Achitecture"* ICCEE-2009, 28-30 December,Dubai,UAE.

Prof. Subarna Bhattacharjee received the M.S. in Computer Science from the DePaul University, Chicago, Illinois, USA, in 1991 and Ph.D. Degrees from the Indian Institute of Technology, Kharagpur, West Bengal, India, in 1997. In 1990, she was a Programmer in Institute of Learning Sciences, North Western University Evanston, Illinois, USA. At present, she is Head of the Department in Computer Science & Engineering, St. Thomas' College of Engineering & Kolkata, West Bengal, India.During 1994-97, she was a Junior Project Officer with the Department of Computer Science & Technology, IIT Kharagpur, India. She has authored or co-authored over 14 journal papers, conference papers. Areas of interest Data, Image and Video Compression.

S. Bhattacharjee, S. Sinha, S. Chattopadhyay and P. Pal Chaudhuri, " Cellular Automata based scheme for Solution of Boolean Equations", Proc. IEE-Comput. Digit.Tech, Volume 143, Number 3, May 1996.
S. Bhattacharjee, J. Bhattacharya, U. Raghavenra, D. Saha and P. Pal Chaudhuri, " A VLSI Architecture for Cellular Automata based Parallel Data Compression ", Proc. VLSI-96 Conference, Bangalore, India, pp. 270-275, January 1996.
S. Bhattacharjee, S. Das, D. Saha, D. Roy Chaudhury and P. Pal Chaudhuri, " A parallel Architecture for Video Compression ", VLSI-97 Conference, Hyderabad, India, pp. 247-252, January 1997.

Mr. Dipak Kumar Kole received the B.Sc degree with Honours in Mathematics and B. Tech. in Computer Science & Engineering from the University of Calcutta, West Bengal, India, in 1998 and 2001, respectively. He obtained the M.Tech. degree in Computer Science & Engineering from the University of Calcutta, West Bengal, India, in 2003. At present he is doing his Ph.D. in the field of VLSI degree from Bengal Engineering & Science University, Shibpur, Howrah.

Mr. Kole is an Asst. Professor of the Department in Computer Science & Engineering, St. Thomas' College of Engineering & Technology, Kolkata, West Bengal, India. He has authored or co-authored over 11 conference papers in area of VLSI, Cryptography and Image Processing.

Dipak K. Kole, Subhadip Basu, "An Automated Group Key Authentication System Using Secret Image Sharing Scheme" International conference on Recent Trends in Information System, IRIS 2006. pp.-98-106, January 6-8, 2006.
H. Rahaman, Dipak K. Kole, Debesh K. Das, Bhargab B. Bhattacharya, "Optimum Test Set for Bridging Fault Detection in Reversible Circuits", In Proc. Asian Test Symp.(ATS 07), Bejing, China, pp. 125-128, 2007.
Dipak K. Kole, H. Rahaman, Debesh K. Das, Bhargab B. Bhattacharya, " A Constructive Algorithm for Synthesis of Reversible Logic Circuits", 12th International Conference on Information Technology (ICIT,09), 2009.

Mr. Amiya Halder received the B.Sc degree with Honours in Physics and B. Tech. in Computer Science & Engineering from the University of Calcutta, West Bengal, India, in 1998 and 2001, respectively. He obtained the M.E. degree in Computer Science & Engineering from the Jadavpur University, West Bengal, India, in 2003. At present, he is doing his Ph.D. in the field of Image Processing from Jadavpur University, Kolkata, West Bengal.

Mr. Halder is a Sr. Lecturer of the Department in Computer Science & Engineering, St. Thomas' College of Engineering & Technology, Kolkata, West Bengal, India. He has authored or co-authored over 10 conference papers in area of Image Processing.

Amiya Halder, Dipak Kumar Kole and Subarna Bhattacharjee, "On-line Colour Image Compression based on Pipelined Architecture," ICCEE-2009, Dubai, UAE, pp.533-537, Dec 28 – 30, 2009.
Amiya Halder, Sandeep Shekhar, Shashi Kant, Musheer Ahmed Mubarki and Anand Pandey " A New Efficient Adaptive Spatial filter for Image Enhancement," ICCEA-2010, pp. 244-46,Bali Island, Indonesi, 26-29 March, 2010.
Amiya Halder, Sourav Dey, Soumyodeep Mukherjee and Ayan Banerjee, "An Efficient Image Compression Algorithm Based on Block Optimization and Byte Compression", ICISA-2010, Chennai, Tamilnadu, India, pp.14-18, Feb 6, 2010.